МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: РОЗРОБКА ВЕБДИЗАЙНУ ТА СТВОРЕННЯ ВЕБСАЙТУ МАРКЕТПЛЕЙСУ

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

(підпис) Мегедя Тараса Тарасовича
(прізвище, ім'я, по батькові)

Науковий керівник (підпис) Грищенко Світлана Миколаївна, канд.пед.наук, ст. досл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми (підпис) Ніжегородцев Владислав Олександрович, канд. пед. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «12» травня 2026 №20

Завідувач кафедри (підпис) Філоненко Михайло Миколайович, канд. фіз.-мат. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Реєстрація № 48 27.05.2026 (дата) (підпис відповідальної особи)

Результати перевірки до захисту (до захисту, на доопрацювання) 12.05.2026р (дата) (підпис)

Результат захисту 4 (добре) (оцінка) 09.06.2026 (дата) (підпис)

Голова комісії: (підпис) О.С. Дюбанов

Члени комісії: (підпис) М.М. Філоненко

(підпис) В.О. Ніжегородцев

(підпис) А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії,секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	22		09.06.2026 Гришенко С.М. Дюбанов О.С. Філоненко М.М. Ніжегородцев В.О. Омельчук А.А. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	7		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	38		

Оцінювання кваліфікаційної роботи науковим керівником – 37 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
75	C	4 (добре)	09.06.2026	Дюбанов О.С. Філоненко М.М. Ніжегородцев В.О. Омельчук А.А. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«01» грудня 2025

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Мегедю Тарасу Тарасовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: «Розробка вебдизайну та створення вебсайту-маркетплейсу»

Затверджена наказом ректора від «28» листопада 2025 № 1526

2. Строк здачі закінченої роботи «9» травня 2026

3. Мета роботи: розробити вебдизайн та створити вебсайт маркетплейс.

Об'єкт дослідження: процес розробки вебдизайну та вебсайту маркетплейсу.

Предмет дослідження: методи, інструменти та технології створення сучасних вебсайтів-маркетплейсів на основі безсерверної архітектури.

Орієнтовна база дослідження: маркетплейс DpuMarket (https://tarasmehed.com).

4. Вихідні дані до роботи: методичні матеріали з розробки веб-сайтів, технічна документація AWS Amplify, Next.js, DynamoDB, AppSync, наукові публікації з UX/UI-дизайну та архітектури e-commerce.

5. Перелік графічного матеріалу: головна сторінка маркетплейсу, сторінка деталей товару, сторінка кошика та оформлення замовлення, особистий кабінет продавця, ER-діаграма бази даних, діаграма архітектури системи, компонентна діаграма, діаграма послідовності, use case, state-machine замовлення.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом):

ВСТУП.

РОЗДІЛ 1. Аналіз предметної області та сучасних технологій розробки вебсайтів-маркетплейсів.

РОЗДІЛ 2. Проєктування вебдизайну та програмної архітектури маркетплейсу.

РОЗДІЛ 3. Програмна реалізація вебсайту маркетплейсу «DpuMarket».

ВИСНОВКИ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 30.10.2025	До 30.10.2025
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	26.12.2025	26.12.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	18.02.2026	18.02.2026
4	**ІІІ розділ** (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	16.04.2026	16.04.2026
5	**Висновки**	01.05.2026	01.05.2026
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	07.05.2026	07.05.2026
7	Отримання відгуку керівника	29.05.2026	29.05.2026
8	Отримання зовнішньої рецензії	01.06.2026	01.06.2026

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи _____ (підпис) Грищенко С.М. (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Ніжегородцев В.О. (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Мегедь Т.Т. (прізвище, ініціали)

АНОТАЦІЯ

«Розробка вебдизайну та створення вебсайту маркетплейсу»

Мета роботи полягає в розробці веб-дизайну та створенні веб-сайту маркетплейсу.
Завданнями роботи є проведення аналізу предметної області електронної комерції; обґрунтування вибору сучасного технологічного стеку для реалізації веб-сайту маркетплейсу; розроблення UX/UI-дизайну маркетплейсу, проєктування моделі даних та механізмів зберігання інформації; програмна реалізація веб-сайту маркетплейсу з повним набором функціональних можливостей;
Предметом дослідження є методи та технології створення сучасних веб-сайтів-маркетплейсів на основі серверлес-архітектури.
Об'єктом дослідження є процес розробки веб-дизайну та програмної реалізації веб-сайтів-маркетплейсів.
Базою дослідження виступає маркетплейс DpuMarket (https://tarasmehed.com).

За результатами дослідження сформульовані методика, етапи та архітектурні рішення розробки сучасного веб-сайту-маркетплейсу. Одержані результати можуть бути використані для практичного запуску маркетплейсу DpuMarket як комерційного продукту, а також як методична основа для розробки інших веб-сайтів-маркетплейсів електронної комерції.

Кваліфікаційна робота містить 72 сторінок, 5 таблиць, 15 рисунків, список використаних джерел 47 найменувань, 2 додатки.

***Ключові слова*:** маркетплейс, електронна комерція, AWS Amplify Gen 2, Next.js 15, Tailwind CSS, UI/UX, веб-сайт.

Рік виконання кваліфікаційної роботи 2026
Рік захисту роботи 2026

ABSTRACT

«Web Design Development and Marketplace Website Creation»

The aim of this work is to develop web design and software implementation of the marketplace.

The tasks of the work **are as follows:** analysis of the e-commerce subject area and the marketplace business model; justification of the choice of a modern technology stack for the marketplace website; development of UX/UI design and architectural design of the system; designing the data model and information storage mechanisms; software implementation of the marketplace website with a complete set of functional capabilities;

The subject of the research **is** methods and technologies for creating modern marketplace websites based on serverless architecture.

The object of the research **is** the process of web design development and software implementation of marketplace websites.

The research is based **on** the DpuMarket marketplace (https://tarasmehed.com).

The results of the research include methodology, stages, and architectural solutions for developing a modern marketplace. The obtained results can be used for the practical launch of the DpuMarket marketplace as a commercial product, as well as a methodological basis for developing other e-commerce marketplace websites.

The qualification work consists of 72 pages, 5 tables, 15 figures, a list of 47 references, and 2 appendices.

Keywords**:** marketplace, e-commerce, serverless architecture, AWS Amplify Gen 2, Next.js 15, Tailwind CSS, web design, web site.

Year of qualification work: 2026

Year of work defense: 2026

ЗМІСТ

ВСТУП..... 4
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-САЙТІВ-МАРКЕТПЛЕЙСІВ..... 7
1.1. Сучасний стан електронної комерції та особливості функціонування маркетплейсів 7
1.2. Огляд сучасних інструментів, технологій і методологій розробки веб-сайтів-маркетплейсів 11
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДИЗАЙНУ ТА ПРОГРАМНОЇ АРХІТЕКТУРИ МАРКЕТПЛЕЙСУ..... 20
2.1. Розробка UX/UI-дизайну та етапи створення інтерфейсу маркетплейсу 20
2.2. Архітектура та проєктування бази даних веб-сайту маркетплейсу..... 30
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-САЙТУ МАРКЕТПЛЕЙСУ «DpuMarket»..... 46
ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
ДОДАТКИ..... 63

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

API – Application Programming Interface – програмний інтерфейс застосунку

AWS – Amazon Web Services – хмарна платформа Amazon

BaaS – Backend-as-a-Service – бекенд як послуга

CDN – Content Delivery Network – мережа доставки контенту

CI/CD – Continuous Integration / Continuous Delivery – безперервна інтеграція та доставка

CSS – Cascading Style Sheets – каскадні таблиці стилів

DynamoDB – Amazon DynamoDB – хмарна NoSQL база даних

GSI – Global Secondary Index – глобальний вторинний індекс

GraphQL – Graph Query Language – мова запитів для API

HTML – HyperText Markup Language – мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

JWT – JSON Web Token – токен на основі JSON

MVP – Minimum Viable Product – мінімально-функціональний продукт

NoSQL – Not Only SQL – нереляційна база даних

REST – Representational State Transfer – архітектурний стиль API

S3 – Amazon Simple Storage Service – хмарне файлове сховище

SEO – Search Engine Optimization – оптимізація для пошукових систем

SQL – Structured Query Language – мова структурованих запитів

SSR – Server-Side Rendering – серверний рендеринг

UI – User Interface – користувацький інтерфейс

URL – Uniform Resource Locator – адреса ресурсу в мережі

UX – User Experience – досвід користувача

WAI-ARIA – Web Accessibility Initiative – Accessible Rich Internet Applications

ВСТУП

Сьогодні світ важко уявити без використання інтернет-технологій, що свідчить про активне розширення бізнесом своїх меж через цифрові канали. Електронна комерція стає основним способом ведення торгівлі в багатьох сегментах, а сучасні маркетплейси формують значну частину цього ринку. За даними Statcounter, понад 70% інтернет-трафіку в Україні припадає на мобільні пристрої, роблячи мобільну адаптивність критично важливою для будь-якого нового веб-сайту електронної комерції. Розробка сучасного маркетплейсу є комплексним завданням, що поєднує проєктування веб-дизайну, програмну реалізацію клієнтської та серверної частин, інтеграцію з хмарними сервісами.

Актуальність даної теми в тому, що електронна комерція є одним із найдинамічніших секторів глобальної економіки, обсяг роздрібних e-commerce продажів у 2024 році перевищив 6 трлн доларів США, а до 2028 року прогнозується зростання до 8 трлн доларів. В Україні e-commerce ринок продовжує зростати навіть в умовах воєнного стану. Попит на нові зручні платформи для купівлі-продажу залишається стабільно високим. Маркетплейси як двосторонні платформи, що об’єднують продавців і покупців, формують понад 65% глобального обсягу e-commerce. Розробка веб-сайту маркетплейсу із застосуванням сучасного серверлес-стеку технологій є актуальним практичним завданням, оскільки забезпечує автоматичне масштабування, низькі стартові витрати та промисловий рівень безпеки, що є критично важливими властивостями для запуску маркетплейсу в умовах конкурентного ринку.

Вчені розглядали дану проблему в своїх роботах[4, 7, 11, 18, 20, 24, 31, 34, 38, 42]. Зокрема, питанням архітектури високонавантажених веб-додатків присвячені роботи М. Клеппманна, патерни UX/UI-проєктування досліджували Я. Нільсен та Б. Фрост, проблематика серверлес-архітектур розкрита у працях інженерів AWS та Vercel, а специфіка електронної комерції в Україні представлена у дослідженнях ринку від Statista, Statcounter та Baymard Institute.

Метою кваліфікаційної роботи є розроблення веб-дизайну та програмна реалізація веб-сайту маркетплейсу DpuMarket із застосуванням серверлес-архітектури на платформі AWS Amplify Gen 2 для забезпечення ефективної взаємодії між продавцями і покупцями в електронній комерції.

Для досягнення мети необхідно вирішити такі завдання:

Проведення аналізу предметної області. Було проаналізовано ринок електронної комерції, визначено типи маркетплейсів, виявлено ключові функціональні вимоги через аналіз конкурентів та персон аудиторії;

Обґрунтування вибору технологічного стеку. У рамках дослідження порівняно сучасні фронтенд-фреймворки (Next.js, Nuxt, Remix, SvelteKit), бази даних (PostgreSQL, MongoDB, DynamoDB), сервіси автентифікації (Auth0, Firebase, Cognito) та бібліотеки компонентів; обрано Next.js 15 + AWS Amplify Gen 2 + shadcn/ui як оптимальне поєднання;

Розроблення UX/UI-дизайну. У рамках дослідження було вивчено сучасні тенденції у дизайні маркетплейсів та розроблено оригінальний інтерфейс маркетплейсу DpuMarket методом design-in-code на основі евристик Я. Нільсена та стандартів Baymard Institute;

Проєктування архітектури та моделі даних. Спроєктовано серверлес-архітектуру з 8 моделями даних (Profile, Product, Category, Like, Rating, Conversation, Message, Order), 9 глобальними вторинними індексами (GSI), декларативною авторизацією через ownerDefinedIn + identityClaim;

Програмна реалізація веб-сайту маркетплейсу. Реалізовано 10 функціональних сторінок (головна, каталог, деталі товару, додавання товару, профіль з 4 вкладками, чат, обрані, пошук, категорія, авторизація) з повним циклом продажу від розміщення товару до отримання замовлення продавцем;

Об’єктом кваліфікаційної роботи є процес розробки веб-дизайну та програмної реалізації веб-сайтів-маркетплейсів.

Предмет дослідження – методи, інструменти та технології створення сучасних веб-сайтів-маркетплейсів на основі серверлес-архітектури.

Методи дослідження – теоретичні – аналіз, узагальнення, систематизація наукової літератури, технічної документації та Інтернет-ресурсів для виділення теоретичних засад дослідження; порівняльний аналіз сучасних веб-технологій, фреймворків та архітектурних патернів; емпіричні – аналіз досвіду провідних маркетплейсів (Amazon, Rozetka, Prom.ua, OLX, Etsy); метод UX-евристичної оцінки за критеріями Я. Нільсена для проєктування інтерфейсу; методи декларативного моделювання даних для NoSQL.

Базою дослідження виступає маркетплейс DpuMarket (https://tarasmehed.com) — повноцінний веб-сайт електронної комерції, розгорнутий на платформі AWS у регіоні eu-central-1 (Франкфурт), що містить каталог товарів, систему категорій, кошик, оформлення замовлень, real-time чат, профіль з управлінням товарами і замовленнями, повністю україномовний інтерфейс.

Наукова-практична значимість одержаних результатів. Розроблений маркетплейс DpuMarket сприяє покращенню користувацького досвіду в українському сегменті електронної комерції, ефективному веденню онлайн-торгівлі для малого та середнього бізнесу, підвищенню конкурентоспроможності за рахунок сучасного дизайну та архітектури, оптимізації бізнес-процесів продавців за рахунок зручних інструментів управління товарами і замовленнями.

Мій особистий внесок у дослідження полягає в самостійному проєктуванні та програмній реалізації маркетплейсу DpuMarket. Особисто розроблено: концепцію та дизайн інтерфейсу методом design-in-code, архітектуру серверлес-системи на AWS Amplify Gen 2

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА СУЧАСНИХ ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ-САЙТІВ-МАРКЕТПЛЕЙСІВ

1.1. Сучасний стан електронної комерції та особливості функціонування маркетплейсів

Електронна комерція є одним із найдинамічніших секторів глобальної економіки. За даними Statista, обсяг роздрібних e-commerce продажів у 2024 році перевищив 6 трлн доларів США, а до 2028 року прогнозується зростання до 8 трлн доларів[33]. Ключовою рушійною силою цього зростання стали маркетплейси – платформи, що об'єднують продавців і покупців в єдиному цифровому просторі. Amazon, Alibaba, eBay, Etsy – ці компанії не просто продають товари; вони є інфраструктурою торгівлі, де десятки мільйонів продавців знаходять своїх покупців.

Маркетплейс як бізнес-модель відрізняється від класичного інтернет-магазину тим, що платформа виступає посередником між продавцями та покупцями, не беручи на себе відповідальності за зберігання товарного запасу. Така модель «двосторонньої платформи» (two-sided platform) має специфічний ефект масштабу: що більше продавців реєструється на платформі – то ширший асортимент вона пропонує покупцям, що, своєю чергою, залучає більше покупців і створює привабливіші умови для нових продавців[20]. Цей «мережевий ефект» є ключовою конкурентною перевагою зрілих маркетплейсів і водночас основним бар'єром для нових гравців при виході на насичений ринок.

За структурою учасників виділяють такі типи маркетплейсів: B2C (Business-to-Consumer) – продавці є компаніями або приватними підприємцями, покупці – фізичні особи (Amazon, Rozetka, Prom.ua); B2B (Business-to-Business) – обидві сторони є юридичними особами (Alibaba B2B, Etsy Wholesale); C2C

(Consumer-to-Consumer) – обидві сторони є фізичними особами (OLX, eBay у класичній моделі); гібридні моделі – поєднання B2C та C2C (більшість сучасних маркетплейсів). Маркетплейс DpuMarket орієнтується на гібридну модель: будь-яка зареєстрована особа може виступати як продавцем, так і покупцем на тій самій платформі.

Фінансова модель маркетплейсу може включати кілька джерел доходу. Комісія з продажів (take rate) – відсоток від кожної угоди, стягується з продавця; для провідних платформ становить 5-15% залежно від категорії. Підписки для продавців – фіксована щомісячна плата за доступ до функцій (просунутої аналітики, пріоритетного розміщення, маркетингових інструментів). Платна реклама всередині платформи – рекламне розміщення для продавців, що прагнуть збільшити видимість своїх товарів. Пов'язані фінансові сервіси – оплата в кредит, страхування угод, факторинг. Для прототипу DpuMarket монетизація не реалізована, проте архітектура продуманий під майбутнє підключення будь-якого з цих механізмів.

В Україні e-commerce ринок демонструє стійке зростання навіть в умовах воєнного стану. Провідними платформами є Rozetka, Prom.ua, OLX, Allo, Foxtrot. Rozetka залишається лідером з часткою понад 30% у сегменті онлайн-ритейлу; Prom.ua позиціонується як b2c-маркетплейс для малого і середнього бізнесу з понад 200 000 магазинів. Специфікою українського ринку є висока частка мобільного трафіку (понад 70% за даними Statcounter[32]), що робить мобільну оптимізацію критично важливою вимогою до нових платформ.

Порівняльний аналіз провідних платформ (Rozetka, Prom.ua, OLX, Amazon, Etsy) проводився методом евристичної оцінки за критеріями юзабіліті Я. Нільсена[24] та з урахуванням рекомендацій Baymard Institute[7] щодо e-commerce UX. У таблиці 1.1 наведено результати порівняння за ключовими критеріями.

Таблиця 1.1

Порівняльний аналіз функціональних можливостей маркетплейсів

Критерій	Rozetka	Prom.ua	OLX	Amazon	DpuMarket
Пошук з фільтрами	✓	✓	✓	✓	✓
Ієрархічні категорії	✓	✓	✓	✓	✓
Зіркові оцінки	✓	✓	✓	✓	✓
Кошик та оформлення замовлення	✓	✓	✗	✓	✓
Вбудований чат	✗	✓	✓	✓	✓
Кабінет продавця	✓	✓	✓	✓	✓
Мобільна адаптивність	✓	✓	✓	✓	✓
Профіль покупця з замовленнями	✓	✓	✗	✓	✓
Real-time повідомлення	✗	✗	✗	✓	✓

Джерело: складено автором на основі аналізу платформ

Основні висновки порівняльного аналізу: всі лідери мають адаптивний мобільний інтерфейс з нижньою навігаційною панеллю; пошук є центральним елементом головної сторінки; checkout-процес скорочений до 1-2 кроків; сторінки товарів містять блок «рекомендованих» товарів. Типові помилки вітчизняних платформ: перевантажений інтерфейс, складна мобільна навігація, відсутність попереднього заповнення форм з профілю при оформленні замовлення, відсутність real-time-повідомлень у чаті.

Аналіз аудиторії маркетплейсу DpuMarket виявив чотири ключові персони (user personas). Персона 1: «Покупець Олег, 32 роки» – офісний працівник, купує різні товари онлайн 2-3 рази на місяць. Основні очікування – швидкий пошук, зрозумілі фільтри, достовірні відгуки, безпечна оплата, зручне відстеження замовлення. Використовує переважно мобільний телефон і робочий ноутбук. Персона 2: «Покупниця Оксана, 26 років» – молода мама в декреті, купує дитячі товари, одяг, косметику, книги. Готова порівнювати ціни, читати детальні огляди. Використовує смартфон, 90% часу проводить у мобільному інтерфейсі. Персона

3: «Продавець Андрій, 40 років» – власник невеликого магазину електроніки. Потребує простих інструментів для розміщення товарів, обробки замовлень, ведення статистики. Використовує ПК більшу частину робочого дня. Персона 4: «Модератор Олена, 28 років» – співробітниця платформи, щодня перевіряє оголошення, реагує на скарги, модерує відгуки. Потребує зручної адміністративної панелі з швидким доступом до критичних функцій.

На основі аналізу персон та конкурентного середовища сформульовано вимоги до маркетплейсу DpuMarket у трьох групах. Функціональні вимоги (Must-have): повний цикл продажу – від розміщення товару до отримання замовлення продавцем; пошук і фільтрація в каталозі; система оцінок (зіркова 1-5); кошик і оформлення замовлення з формою доставки; профіль покупця з вкладками Продажі і Покупки; чат між учасниками з real-time-повідомленнями; список обраних товарів; завантаження зображень товарів і аватарів. Нефункціональні вимоги: завантаження ключових сторінок до 2 секунд; mobile-first адаптивний дизайн; SSL-захист та безпечне зберігання даних користувачів. Проєктні обмеження: одноосібна розробка у форматі бакалаврської роботи; нульові або мінімальні стартові інфраструктурні витрати.

Окремої уваги заслуговує питання локалізації. Більшість успішних маркетплейсів, що орієнтуються на конкретний регіон, повністю локалізують свій інтерфейс – не лише переклад текстів, а й адаптацію форматів дат, валюти, номерів телефонів, адресних форм під місцеві стандарти. Маркетплейс DpuMarket реалізовано повністю українською мовою, що є свідомим рішенням, спрямованим на українського споживача й відповідає сучасному тренду підтримки україномовних цифрових продуктів.

Висновки до підрозділу 1.1. Маркетплейси є домінуючою моделлю e-commerce із суттєвою присутністю в Україні. Визначено ключові функціональні блоки (пошук, оцінки, чат, замовлення, мобільна адаптивність) та чотири персони аудиторії. Порівняльний аналіз конкурентів виявив кращі практики і типові прогалини вітчизняних платформ. Сформульовано функціональні та

нефункціональні вимоги до маркетплейсу DpuMarket, які є базою для подальшого проєктування.

1.2. Огляд сучасних інструментів, технологій і методологій розробки веб-сайтів-маркетплейсів

Серверна частина (backend) сучасного веб-сайту може бути реалізована двома принципово різними способами. Класичний шлях – виділений сервер з власним застосунком та базою даних, які розробник налаштовує й адмініструє вручну. Сучасний шлях – серверлес-архітектура, у якій маркетплейс працює на готовій хмарній платформі, а власник продукту платить лише за реально спожиті ресурси й узагалі не керує серверами. Цей зсув парадигми відбувся поступово протягом 2015-2025 років, і сьогодні практично всі нові маркетплейси будуються саме на хмарних/серверлес-підходах[34].

Важливо розуміти, що це не просто інженерна мода, а виразний галузевий тренд. За останні п'ять років практично всі найбільші маркетплейси світу так чи інакше перейшли на серверлес-архітектуру або інтегровану хмарну BaaS-платформу (Backend-as-a-Service). На відповідних архітектурах, цілком або частково, працюють Amazon, Airbnb, Netflix, Shopify, eBay, Etsy, Stripe, Vinted, Bolt, Glovo[34]. Для нових маркетплейсів, які запускаються сьогодні, серверлес взагалі є замовчуванням.

Чому ж відбувся цей зсув? Тому що серверлес дає маркетплейсу кілька конкретних, видимих для бізнесу переваг. Перша – маркетплейс витримує піки трафіку без падінь. У дні великих розпродажів («Чорна п'ятниця», «Кіберпонеділок», «11.11») трафік може зрости у 10-50 разів за лічені хвилини. На класичному сервері це майже гарантоване «падіння» сайту. На серверлес-платформі система автоматично запускає стільки ресурсів, скільки потрібно. Друга перевага – швидкість виходу на ринок (Time-to-Market). Замість написання систем реєстрації, відновлення пароля, керування ролями, завантаження зображень, розсилки

листів – маркетплейс отримує всі ці компоненти готовими з BaaS-платформи. Третя – оплата пропорційно реальному навантаженню (pay-as-you-go). Четверта – глобальна швидкість через CDN. П'ята – безпека вже вбудована провайдером[34].

Розглянемо ці переваги детальніше. Перша й найважливіша – маркетплейс витримує піки трафіку без падінь. У дні великих розпродажів («Чорна п'ятниця», «Кіберпонеділок», «11.11», запуск нової колекції) трафік маркетплейсу може зрости у 10–50 разів за лічені хвилини. На класичному сервері це майже гарантоване «падіння» сайту, бо потужностей наперед виділено фіксовано. На серверлес-платформі система автоматично запускає стільки ресурсів, скільки потрібно (auto-scaling до тисяч паралельних екземплярів за секунди), а коли пік минає – повертається до базового рівня.

Друга перевага – швидкість виходу на ринок (Time-to-Market). Замість написання з нуля систем реєстрації, відновлення пароля, керування ролями, завантаження зображень, розсилки листів, маркетплейс отримує всі ці компоненти готовими з BaaS-платформи. За оцінками AWS, типовий e-commerce проєкт скорочує перший реліз на 40–60% порівняно з традиційним стеком, а команда може зосередитися на бізнес-логіці (модель товарів, ціноутворення, UX) замість інфраструктурного коду.

Третя перевага – вартість зростає пропорційно реальному навантаженню (pay-as-you-go). Власнику маркетплейсу не потрібно наперед платити за серверні потужності «на майбутнє»: поки немає продажів – платіж близький до нуля, з'являється трафік – платіж зростає лінійно з доходом. Це принципово знижує ризик нового проєкту: маркетплейс на старті може обслуговувати десятки користувачів за $5/місяць, а при зростанні до десятків тисяч – платити $500, без перебудови архітектури.

Четверта перевага – глобальна швидкість для користувачів у будь-якій точці світу. Сторінки маркетплейсу попередньо генеруються (SSG/ISR) і розповсюджуються через мережу CDN-серверів (Amazon CloudFront має понад

600 точок присутності у понад 100 містах). Замість одного запиту до серверу за тисячі кілометрів – користувач отримує контент із найближчого CDN-вузла за десятки мілісекунд, що покращує метрики Web Vitals (LCP, FID, CLS) і прямо впливає на SEO-позиції та conversion rate.

П'ята перевага – безпека вже вбудована в платформу. Хешування паролів за сучасними алгоритмами (bcrypt, scrypt), захист від brute-force, ротація ключів автентифікації, відповідність вимогам GDPR і PCI DSS, регулярні безпекові оновлення – все це робить хмарний провайдер. Малому стартапу або одноосібному розробнику фактично неможливо самостійно досягти такого ж рівня безпеки на класичному сервері.

Серверлес-підхід має й об'єктивні мінуси, які варто враховувати. Найвідоміший – «холодний старт» (cold start): функція, яка певний час не використовувалася, потребує більше часу на перший запит (для Node.js Lambda – 200–500 мс). Для маркетплейсу з постійним трафіком це не критично, бо функції залишаються «теплими», але для рідко вживаних сценаріїв затримка помітна. Другий мінус – певна прив'язка до конкретного хмарного провайдера (vendor lock-in): міграція з AWS на Google Cloud вимагатиме переписування коду. Цей ризик частково компенсується тим, що архітектурні принципи (GraphQL API, NoSQL, JWT) переносяться, а специфічними є переважно конфігураційні файли.

Для повноти картини варто згадати кілька суміжних архітектурних концепцій. JAMstack (JavaScript, APIs, Markup) – підхід, у якому сторінки фронтенду попередньо генеруються як статичні файли і розповсюджуються через CDN; це підмножина серверлес-підходу для контентних сайтів. Edge computing – виконання серверного коду не в центральному дата-центрі, а на CDN-вузлах ближче до користувача (Cloudflare Workers, AWS Lambda@Edge). Micro-frontends – архітектура, у якій великий веб-сайт ділиться на незалежно розгортувані частини; для проєкту масштабу бакалаврської роботи це надлишково, але для великих маркетплейсів типу Amazon це фактичний стандарт.

Для маркетплейсу окремо важливою є SEO-оптимізація: пошукові системи Google і Bing віддають перевагу сайтам з повноцінним HTML-контентом, що генерується на сервері (SSR) або заздалегідь (SSG/ISR). На відміну від традиційних SPA, де перший запит повертає порожню сторінку, серверлес-фреймворки на кшталт Next.js забезпечують hybrid-rendering: критичні сторінки (каталог, картка товару, лендінг) генеруються заздалегідь і кешуються в CDN, а інтерактивні (особистий кабінет, кошик) працюють як клієнтські компоненти. За даними Google Search Central, час до повного відображення сторінки (LCP) понад 2,5 секунди суттєво погіршує ранжування, тому архітектурні рішення на користь SSR/SSG/ISR прямо впливають на конверсію та видимість маркетплейсу в пошуку.

Сучасна тенденція у фронтенд-розробці — використання React Server Components (RSC), запроваджених у React 18 і повноцінно підтриманих у Next.js 13+. Принципова відмінність RSC від традиційних компонентів у тому, що вони виконуються виключно на сервері й не надсилають JavaScript у браузер: клієнт отримує вже згенерований HTML, що суттєво зменшує розмір бандла та покращує метрики продуктивності. Для маркетплейсу це означає, що статичні частини сторінок (header, footer, метадані, навігація) реалізуються як RSC, тоді як інтерактивні елементи (кошик, фільтри, чат) — як 'use client' компоненти. Такий гібридний підхід зменшує JavaScript-бандл на 40–60% порівняно з повністю клієнтським рендерингом, що особливо критично для мобільних користувачів з повільним інтернетом.

Окреме питання – керування глобальним станом у клієнтських компонентах маркетплейсу. Аналіз показав, що для маркетплейсу масштабу DpuMarket найдоцільніше використовувати комбінацію: React Context для трьох ключових контекстів (Auth, Likes, Cart), а серверний стан синхронізувати через TanStack Query або вбудовані механізми AWS Amplify Data.

Не менш важливим є питання обробки зображень товарів — критичний компонент будь-якого маркетплейсу. Завантаження великих оригінальних фото

з мобільного телефону (3–8 МБ) напряму на сторінку каталогу категорично неприпустиме: на 3G-з'єднанні таке завантаження зайняло б десятки секунд. Стандартним рішенням є генерація кількох розмірів (thumbnail 200px, medium 600px, large 1200px) на стороні сервера через AWS Lambda або CloudFront Image Optimization, а також використання сучасних форматів (WebP, AVIF) із автоматичним fallback до JPEG. Next.js надає вбудований компонент <Image>, що автоматизує цю задачу: lazy-loading при прокручуванні, responsive srcSet, blur-placeholder під час завантаження. Для маркетплейсу DpuMarket обрано підхід зі збереженням оригіналу в S3.

Серед провідних BaaS-платформ для практичної реалізації обрано AWS Amplify. Головна причина – AWS є найбільшою хмарною платформою зі 47% ринку (за Statista 2024), і саме на її інфраструктурі, в тій чи іншій формі, працюють Amazon, Airbnb, Netflix, Pinterest, Lyft, Slack, Zoom, eBay, Stripe, Shopify, Bolt[34]. Звідси – найбагатша екосистема готових сервісів, найбільша спільнота розробників, найкраща документація та найдовший досвід обслуговування високонавантажених проєктів. AWS Amplify Gen 2 є другою генерацією платформи: замість JSON-конфігурацій (Amplify Gen 1) всі ресурси описуються через TypeScript, що забезпечує суворе типування, рефакторинг і підтримку IDE.

Одним з ключових компонентів маркетплейсу є система керування даними. Для серверлес-архітектур стандартом стало використання повністю керованих NoSQL-баз: Amazon DynamoDB, Google Firestore, MongoDB Atlas. Реляційні бази (PostgreSQL, MySQL) залишаються актуальними для складних транзакційних систем, але для маркетплейсу, де більшість операцій – це точкові запити (get by id, filter by owner, list with status), NoSQL забезпечує кращу масштабованість при нижчих витратах. Ключова особливість DynamoDB – модель «single-table design», де всі сутності зберігаються в одній таблиці, або «multi-table» з окремою таблицею для кожної моделі (підхід Amplify Data). Обидва підходи забезпечують O(1) латентність при правильному виборі ключів доступу[2].

Важливим інструментом ефективних запитів у DynamoDB є глобальні вторинні індекси (GSI). GSI – це фактично автоматично синхронізована копія таблиці з іншим partition key, що дозволяє виконувати ефективні запити за будь-яким полем, не тільки за первинним ключем. Наприклад, GSI з partition key = userId дозволяє за одним запитом отримати всі товари або всі замовлення конкретного користувача, уникаючи повного скану таблиці. Ціна GSI – приблизно подвоєння витрат на зберігання та запити по індексованих даних, що є прийнятним компромісом для маркетплейсу[2].

Для API-шару (взаємодія між клієнтом та базою даних) розглядалося два підходи: REST API та GraphQL. REST API є класичним підходом: кожен ресурс має власний endpoint (GET /products, POST /orders, PATCH /products/:id). GraphQL – більш сучасний підхід: клієнт формулює запит саме з тими полями, які йому потрібні, не отримуючи зайвих даних (over-fetching) і не виконуючи кількох запитів для отримання пов'язаних даних (under-fetching/N+1 problem). Для маркетплейсу, де мобільний клієнт часто хоче отримати лише назву, ціну і зображення товару (без опису, SKU, ownerId), GraphQL природно вирішує проблему ефективності трафіку. AWS AppSync автоматично генерує GraphQL-схему та резолвери на основі декларативних моделей Amplify Data, значно скорочуючи обсяг ручного коду[5].

Окремо варто розглянути протокол WebSocket, що використовується для real-time-комунікації в чаті. HTTP є однонаправленим: клієнт ініціює запит – сервер відповідає. Це підходить для більшості операцій (отримати список товарів, розмістити замовлення), але катастрофічно неефективно для чату: клієнт змушений постійно опитувати сервер (polling) на наявність нових повідомлень. WebSocket встановлює постійне двостороннє з'єднання, яке дозволяє серверу «проштовхувати» нові дані клієнту в режимі реального часу. AWS AppSync реалізує GraphQL Subscriptions через механізм MQTT over WebSocket – відкрита WebSocket-сесія автоматично доставляє нові записи при виконанні мутації, що відповідає умові фільтру підписки.

Для побудови компонентного інтерфейсу маркетплейсу розглядалися декілька підходів. «Atomic design» – методологія Бреда Фроста[11], що пропонує ієрархію: атоми (кнопки, інпути) → молекули (форма пошуку) → організми (хедер) → шаблони → сторінки. Ця методологія є концептуальною основою більшості сучасних дизайн-систем. «Design tokens» – підхід, де значення дизайну (кольори, шрифти, відступи, радіуси) виносяться у змінні, що використовуються однаково у дизайнерському інструменті і в коді. У DpuMarket design tokens реалізовані через CSS custom properties (var(--primary), var(--border-radius)) у файлі globals.css. «Utility-first CSS» – Tailwind CSS надає набір атомарних CSS-класів, що комбінуються прямо в HTML/JSX, замість написання окремих CSS-правил. Дослідження за 2024 рік (State of CSS[31]) показують, що Tailwind CSS є найпопулярнішою CSS-методологією серед розробників з показником 80%+ обізнаності та 60%+ регулярного використання.

Таблиця 1.2

Порівняльний аналіз фронтенд-фреймворків для маркетплейсу

Критерій	Next.js 15	Nuxt 3	Remix	SvelteKit
SSR / SSG / ISR	✓ / ✓ / ✓	✓ / ✓ / ✗	✓ / ✗ / ✗	✓ / ✓ / ✗
React-компоненти	✓ (React 19)	✗ (Vue 3)	✓	✗ (Svelte)
AWS Amplify офіц. підтримка	✓ повна	часткова	часткова	відсутня
Спільнота та документація	дуже велика	велика	середня	середня
Turbopack (швидка збірка)	✓ вбудований	✗	✗	✗

Джерело: складено автором

Таблиця 1.2 демонструє, що Next.js 15 є єдиним фреймворком із повним набором критичних для маркетплейсу можливостей. Ключовими диференціаторами є офіційна підтримка AWS Amplify та вбудований Turbopack. Nuxt 3 та SvelteKit використовують альтернативні екосистеми (Vue та Svelte відповідно), що означає несумісність з Amplify React-компонентами та значно меншу бібліотеку готових UI-рішень для e-commerce.

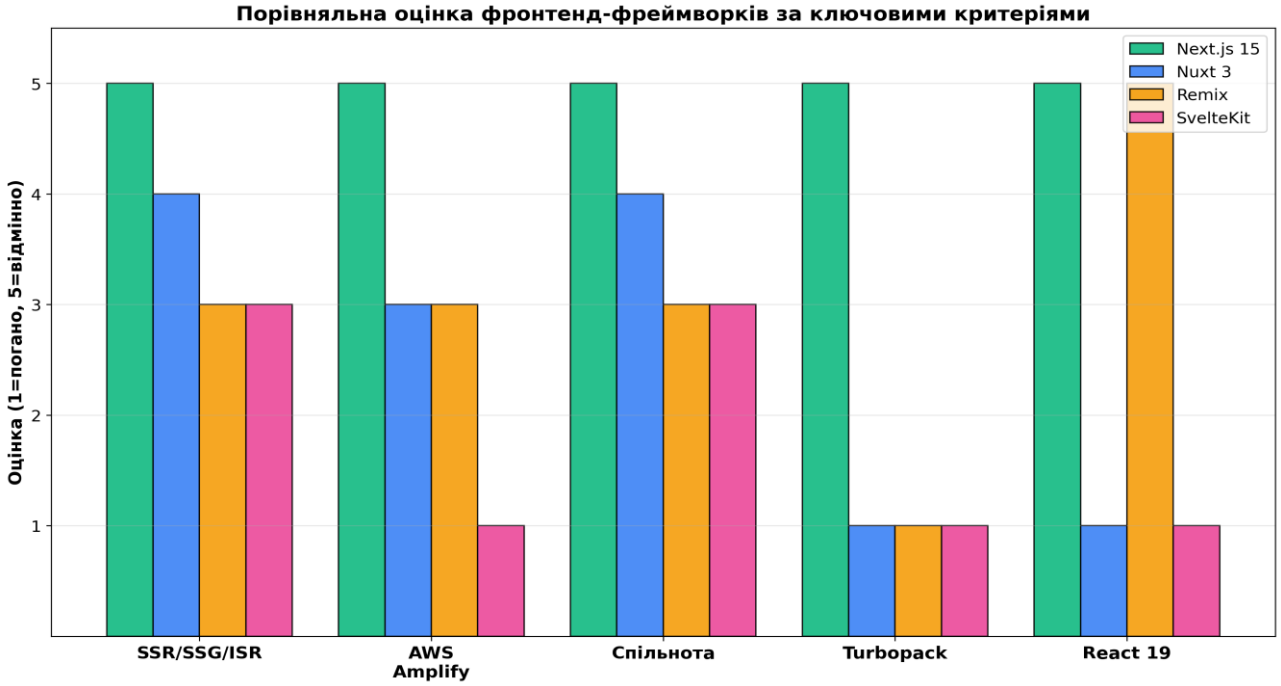


Рис. 1.1. Порівняльна оцінка фронтенд-фреймворків за п'ятьма ключовими критеріями

Джерело: складено автором за результатами аналізу

Для автентифікації порівнювалися Auth0, Firebase Authentication та Amazon Cognito. Auth0 – найбагатший за функціями, але дорогий при масштабуванні (ціна по MAU – monthly active users). Firebase Authentication – безкоштовний до певного порогу, але прив'язаний до Google Firebase, не інтегрований з AWS і вимагає окремої логіки авторизації запитів до DynamoDB. Amazon Cognito – безкоштовний до 50 000 MAU, нативно інтегрований з AWS Amplify та AppSync, підтримує OAuth 2.0/OIDC, User Pool Groups для рольової моделі, SAML/OIDC identity federation. Cognito надає SDK для серверного доступу (для SSR через cookie-based session), що є критично важливим для Next.js App Router.

Для бібліотеки компонентів порівнювалися: Material UI (Google Material Design, найпоширеніша, але характерна і важка – близько 350 KB); Ant Design (потужна, але орієнтована на B2B-застосунки, із виразним корпоративним дизайн-стилем); shadcn/ui (не є бібліотекою у традиційному сенсі – компоненти копіюються в проєкт, нуль runtime-залежностей, повний контроль над кодом,

TypeScript-first). Для маркетплейсу, де дизайн має бути унікальним і повністю контрольованим, shadcn/ui є оптимальним вибором: відсутність runtime-залежностей після встановлення унеможливлює поломку від оновлень бібліотеки, а Tailwind-базована стилізація ідеально поєднується з загальною дизайн-системою проєкту.

Остаточний технологічний стек маркетплейсу DpuMarket: Next.js 15 (React 19, TypeScript strict) – фронтенд-фреймворк; AWS Amplify Gen 2 – оркестрація бекенду і провіжинг AWS-ресурсів; AWS AppSync GraphQL – API-шар з real-time subscriptions через WebSocket; Amazon DynamoDB – NoSQL БД у регіоні eu-central-1 (Франкфурт); AWS Lambda – кастомна бізнес-логіка (Lambda `search-products` і `index-product`); Amazon Cognito – автентифікація з email, CODE-верифікацією, парольною політикою; Amazon S3 – файлове сховище й розподілена доставка зображень; Tailwind CSS 4 (@tailwindcss/postcss) – CSS-фреймворк; shadcn/ui (new-york style) – дизайн-система; Radix UI – accessible-примітиви; embla-carousel-react – карусель; lucide-react – іконки; Inter + Geist Mono (next/font) – шрифти; AWS Amplify Hosting – розгортання. Менеджер залежностей – npm із прапором --legacy-peer-deps. Версійний контроль здійснюється через — GitHub.

Висновки до розділу 1. У першому розділі проведено комплексний аналіз предметної області та технологій. Показано, що маркетплейси є домінуючою бізнес-моделлю e-commerce зі стійким зростанням в Україні та у світі. Визначено функціональні вимоги та проведено конкурентний аналіз. Досліджено серверлес-парадигму як провідну архітектурну тенденцію з п'ятьма ключовими перевагами для маркетплейсу. Проведено порівняльний аналіз фронтенд-фреймворків (Next.js переміг за сукупністю критеріїв), баз даних (DynamoDB), сервісів автентифікації (Cognito), та бібліотек компонентів (shadcn/ui). Обрано технологічний стек Next.js 15 + AWS Amplify Gen 2, що є предметом практичної реалізації у наступних розділах.

РОЗДІЛ 2

ПРОЄКТУВАННЯ ВЕБ-ДИЗАЙНУ ТА ПРОГРАМНОЇ АРХІТЕКТУРИ МАРКЕТПЛЕЙСУ

2.1. Розробка UX/UI-дизайну та етапи створення інтерфейсу маркетплейсу

Проєктування інтерфейсу маркетплейсу DpuMarket будувалося на підході design-in-code (code-first design): дизайнерські рішення ухвалювалися й одразу впроваджувалися засобами Tailwind CSS та shadcn/ui без окремого прототипування у спеціалізованих інструментах. Такий підхід є поширеним у невеликих командах та при розробці MVP, де швидкість виходу на ринок важливіша за ретельне попереднє проектування макетів. Ключова перевага – цикл «ідея → реалізація → перевірка у браузері» скорочується з днів до хвилин, що дозволяє значно частіше перевіряти гіпотези безпосередньо на реальному пристрої.

Перш ніж переходити до прототипування інтерфейсу, було сформовано чотири персони цільової аудиторії маркетплейсу. Перша персона — Покупець-новачок (вік 18-30, мобільний користувач, шукає швидкі покупки за низькою ціною, чутливий до зручності пошуку та фільтрації). Друга персона — Покупець-експерт (вік 30-50, користується десктопом, порівнює характеристики, читає відгуки, очікує детальну інформацію про товар). Третя персона — Дрібний продавець (фізична особа, продає речі з власних активів, потребує простий інтерфейс розміщення товарів без складних форм). Четверта персона — Малий бізнес (юридична особа, продає товари серійно, потребує функціонал керування каталогом і відстеження продажів).

На основі персон сформульовано основні користувацькі сценарії (user stories): як Покупець, я хочу знайти товар у каталозі за назвою, категорією та ціною, щоб швидко обрати потрібне; як Покупець, я хочу переглянути детальну

сторінку товару з фотографіями, описом та оцінками, щоб ухвалити рішення про купівлю; як Покупець, я хочу поспілкуватися з продавцем у чаті, щоб уточнити нюанси доставки; як Продавець, я хочу швидко додати новий товар до каталогу з кількома фотографіями, щоб мінімізувати час публікації; як Продавець, я хочу бачити отримані замовлення та керувати їх статусом, щоб контролювати процес продажу. Ці сценарії безпосередньо мапуються на use case-и системи, представлені на рис. 2.1.

Use Case діаграма використовується як інструмент для візуалізації функціональних вимог системи з точки зору акторів (зовнішніх користувачів і систем). У контексті маркетплейсу DpuMarket виділено три актори: Покупець (споживач товарів), Продавець (постачальник товарів) та Система (фоновий обробник, що виконує пошукові операції через AWS Lambda). Перетинальні use case-и між акторами (як-от «Реєстрація / Вхід», «Профіль користувача») вказують на спільну функціональність, доступну обом ролям після автентифікації.

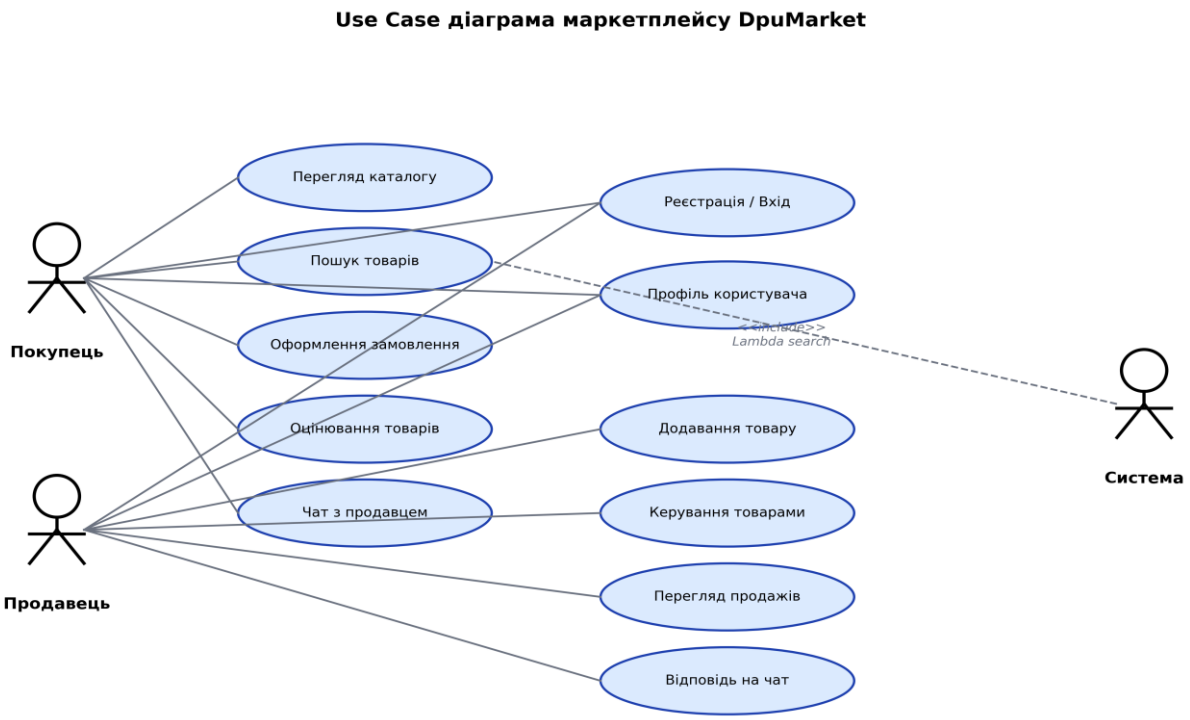


Рис. 2.1. Use Case діаграма маркетплейсу DpuMarket

Джерело: розроблено автором

Методологічною основою проєктування стали десять евристик юзабіліті Якоба Нільсена[24] та стандарти Baymard Institute для e-commerce[7]. Евристики Нільсена системно визначили конкретні рішення: видимість стану системи – спінери завантаження Loader2 на всіх async-операціях, миттєвий оптимістичний стейт лайку до підтвердження сервера; відповідність реальному світу – іконки кошика (ShoppingCart), серця-лайку (Heart), зірок рейтингу (Star) із загальновідомим семантичним значенням; контроль і свобода – кнопка «Скасувати» у всіх формах, закриття будь-якого модального вікна через Escape чи overlay; послідовність – єдині компоненти Button, Input, Card, Dialog на всіх 10 сторінках без виключень; запобігання помилкам – клієнтська валідація форм перед відправкою з детальними підказками; розпізнавання замість запам'ятовування – категорії видимі в меню та в breadcrumb сторінки товару, не потребують запам'ятовування шляху; гнучкість – фільтри на сторінці пошуку і каталогу; мінімалізм

– акцент на товарах без зайвих рекламних банерів у каталозі; допомога в помилках – зрозумілі повідомлення при провалах автентифікації з конкретними порадами.

Дослідження Baymard Institute[7] показують, що типовий e-commerce сайт має 67 критичних UX-проблем у checkout-процесі, фільтрації та навігації. Це визначило кілька конкретних рішень у DpuMarket: pre-fill форми замовлення з профілю користувача (скорочує час checkout); сітка товарів 2 колонки на мобільному (оптимальна за Baymard для перегляду великим пальцем); максимально компактна картка товару з видимою ціною і кнопкою «Додати» без скролу; відображення наявності (In Stock / Out of Stock) прямо на картці без переходу на деталі.

Типографіка DpuMarket заснована на двох шрифтах. Основний – Inter: нейтральний гротескний шрифт з оптимізованими метриками для екранів, підтримує кириличний алфавіт, завантажується через `next/font/google` із підключенням через CSS-змінну `--font-inter`. Моноширний – Geist Mono: для артикулів SKU, ідентифікаторів замовлень, технічних кодів, завантажується через `--font-geist-mono`. Обидва шрифти рендеруються з `swap`-стратегією через next/font і не спричиняють FOUT (Flash of Unstyled Text), оскільки Next.js автоматично додає `<link rel="preload">` для них у HTML-заголовку. Шкала розмірів визначена Tailwind-утилітами: text-xs (12px), text-sm (14px), text-base (16px), text-lg (18px), text-xl-5xl для заголовків. Рядковий інтервал – leading-relaxed (1.625) для тіла тексту, leading-tight (1.25) для заголовків карток.

Сітка розмітки (layout grid) побудована на принципі «responsive container». Основний контейнер обмежений `max-w-7xl` (1280px) і центрований через `mx-auto`. Горизонтальні відступи адаптивні: `px-4` (16px) на мобільних, `sm:px-6` (24px) на планшетах, `lg:px-8` (32px) на десктопах. Сітка карток товарів: `grid-cols-2` на мобільних – дві картки поряд, найприродніша модель для перегляду великим пальцем; `sm:grid-cols-2` (рівно 2 для планшетів у портретній орієнтації), `md:grid-cols-3`, `lg:grid-cols-4`, `xl:grid-cols-5` – прогресивне розширення.

Baymard Institute[7] рекомендує 3-4 колонки на десктопі як оптимальні для e-commerce; DpuMarket розширює до 5 колонок на дуже широких екранах для кращого охоплення асортименту. Усі відступи між елементами кратні 4px (базова одиниця Tailwind spacing scale).

Компонентна архітектура базується на принципі «headless + styled»: Radix UI надає поведінку без стилів (keyboard navigation, focus trap, aria-attributes, portal mounting, escape-to-close, outside click detection), Tailwind CSS надає стилі без JS. shadcn/ui поєднує ці два рівні в готові до використання TypeScript-компоненти: Button (6 варіантів: default, outline, ghost, secondary, destructive, link; 5 розмірів: default, sm, lg, icon, icon-sm), Input, Textarea, Card, Dialog, Drawer, Accordion, Slider, Carousel. Рішення принципово відрізняється від монолітних бібліотек (Material UI ~350KB, Ant Design ~500KB), де поведінка і стилі нерозривно переплетені. Після копіювання shadcn-компонентів у `/src/components/ui/` – нуль runtime-залежностей від зовнішніх бібліотек компонентів.

Навігаційна архітектура реалізована у двох взаємовиключних зонах. Хедер (desktop, md+): логотип-SVG (`/favicon.svg`, векторний текст «DpuMarket»), рядок пошуку з live-дебаунсом 300 мс і `router.replace` (щоб не клоткувати history), навігаційні іконки Home, CirclePlus (додати), MessageCircle (чат), Heart (обрані), ShoppingCart зі значком-badge кількості товарів, UserRound (профіль). Кнопка-гамбургер відкриває Drawer зі списком категорій із DynamoDB. Bottom navigation bar (mobile, hidden md+): фіксована `fixed bottom-0 z-40`, 5 іконок рівномірно по ширині (Home, ShoppingCart з badge, CirclePlus, MessageCircle, UserRound), `paddingBottom: env(safe-area-inset-bottom)` для iPhoneX+ з notch і Dynamic Island. Відповідно до дослідження Nielsen Norman Group, bottom navigation скорочує час навігації на мобільних до 1-2 кроків для 90% частих сценаріїв.

Реалізація головної сторінки (`/`) включає hero-карусель та нескінченну сітку товарів. Карусель на основі embla-carousel-react відображає три рекламні банери (`/banners/summer-one.jpg`, `summer-two.jpg`, `summer-three.jpg`) з автопрокруткою кожні 5 секунд, індикаторами слайдів і навігаційними кнопками

Previous/Next (видимі при hover). Нескінченний скрол (infinite scroll) реалізований через IntersectionObserver: sentinel-елемент внизу сітки спостерігається; при перетині viewport завантажується наступна сторінка через `nextToken` з DynamoDB (limit = 12 товарів). Після завантаження всіх товарів відображається підсумкове повідомлення про кількість.

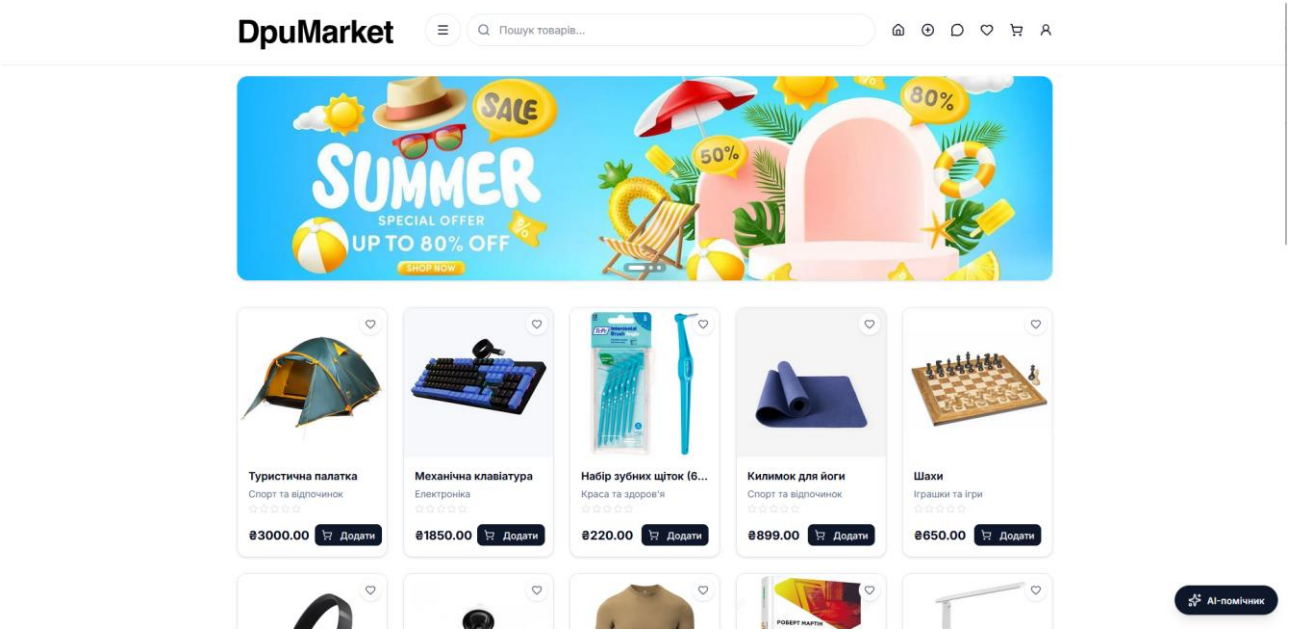


Рис. 2.2. Головна сторінка маркетплейсу DpuMarket (скриншот реалізації)

Джерело: розроблено автором

Картка товару (ProductCard) реалізує комплекс UX-рішень. Квадратна пропорція зображення (aspect-square) гарантує однорідну висоту сітки незалежно від розміру завантаженої картинки. Тег категорії відображається у лівому верхньому куті зображення (badge). Кнопка лайку у правому верхньому куті видима лише при hover (opacity-0 → opacity-100 при group-hover), що не захаращує UI при пасивному перегляді. Оптимістичне оновлення лайку: серце заповнюється fill-red-500 миттєво до підтвердження сервера, а у випадку помилки відкочується – патерн, що використовується в Instagram, Twitter, Airbnb для відчуття миттєвої реакції. Кнопка «Додати» (ShoppingCart) тимчасово замінюється на «Додано» (Check) на 1.5 секунди після натискання з паралельним відкриттям CartDrawer. Зіркові оцінки підтримують дробові значення через CSS clip у компоненті StarRating: fillPct = Math.max(0, Math.min(1, display - (i-1))) * 100.

Сторінка деталей товару (`/product/[id]`) використовує двоколонну сітку (md:grid-cols-2) на десктопі і однорядне вертикальне розміщення на мобільному. Зображення – повний квадрат з `fill` і `object-cover` через компонент `<S3Image />`. Над зображенням – посилання-категорія (Link), що провадить до `/category/{slug}` відповідної категорії. Ціна – виділена шрифтом 3xl bold. Статус наявності – зелений badge «В наявності (N)» або червоний «Немає в наявності». Лічильник кількості (−/+) обмежений зверху реальним stock товару. Кнопка «Додати до кошика» – full-width на мобільному, flex-1 на десктопі. Блок зіркового оцінювання для автентифікованих користувачів (вибір 1-5 зірок через hover + click, з update замість create при повторному голосуванні). Захисні картки («Безкоштовна доставка», «Безпечна оплата», «Легке повернення») у нижній частині.

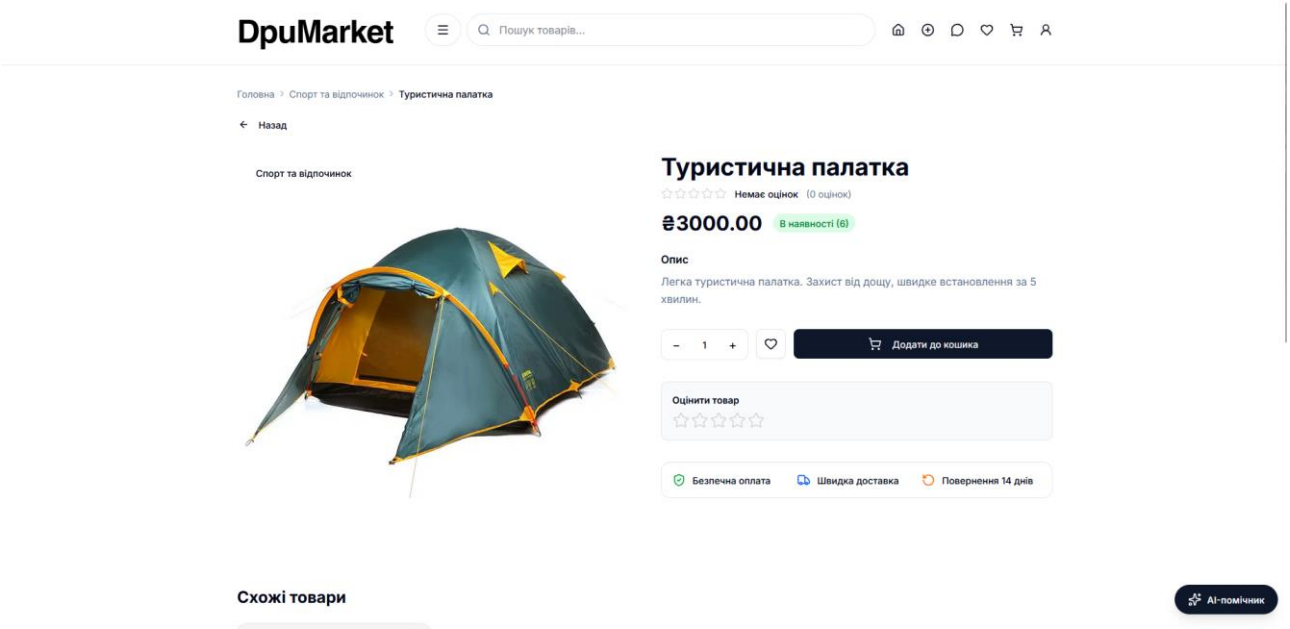


Рис. 2.3. Сторінка деталей товару з оцінюванням і кошиком (скриншот реалізації)

Джерело: розроблено автором

Компонент `FilterableProductGrid` реалізує in-memory фільтрацію та сортування без мережевих запитів. Фільтри на горизонтальній панелі зверху: діапазон ціни (Min/Max input з `type="number" inputMode="numeric"`), діапазон кількості на складі (Min/Max), сортування (select: Найновіші / Ціна від низької до високої

/ Ціна від високої до низької). Фільтри ціни та кількості приховані на мобільних (`hidden sm:flex`) – iOS auto-zoom при фокусі на number-input і обмежений простір унеможливлюють зручне використання. Текстовий запит з URL-параметру `?q=` синхронізується з `useEffect([initialQuery])`. Рахунок знайдених товарів відображається з правильним відмінюванням (1 «товар», 2-4 «товари», 5+ «товарів»). Результати завжди повністю рендеруються (no pagination) – для prototype-масштабу це допустимо.

Кошик (CartDrawer) відкривається з правого боку як Drawer-компонент, шириною `max-w-sm` (384px). Список товарів із S3Image-мініатюрами, кнопками ±, видаленням і рядковою ціною. Підсумок і кнопка «Перейти до оформлення» у sticky-footer drawer. CheckoutModal – Dialog з прокруткою `max-h-[90vh]`, що підтримує довгі форми на мобільних. При успішному замовленні сторінка замінюється success-станом з іконкою CheckCircle2 та кнопкою «Продовжити покупки». Після успішного checkout кошик очищується через `clearCart()` і стан у localStorage обнуляється.

Профіль користувача реорганізований у чотири вкладки з badge-лічильниками. Вкладки реалізовані власним CSS без зовнішніх бібліотек: `sticky top-0 z-10` для фіксації при прокрутці, `bg-background border-b` для відокремлення, індикаторний рядок `h-0.5 bg-primary absolute bottom-0` для активної вкладки. На мобільних іконка+текст розміщені вертикально (`flex-col gap-1 text-[11px]`) для вміщення 4 вкладок на 320px-екрані; badge-лічильник – абсолютно позиційований `-top-1.5 -right-2.5` поверх іконки. На десктопі – іконка+текст горизонтально з badge поруч з текстом. Вкладка «Продажі» показує замовлення від покупців (ordersBySeller) з кнопками управління статусами: Підтвердити (PENDING→CONFIRMED), Відправлено (CONFIRMED→SHIPPED), Виконано (SHIPPED→COMPLETED), Скасувати. Вкладка «Покупки» – лише перегляд власних замовлень (ordersByBuyer).

Сторінка чату реалізована у стилі популярних месенджерів: двопанельне компонування (список розмов + активний діалог), на мобільних – перемикання

між панелями. Бульки повідомлень мають градуйовані радіуси: якщо поруч з попереднім повідомленням того ж відправника – `rounded-2xl rounded-tr-md` (відтятий кут), якщо останнє в групі – `rounded-2xl rounded-br-md` (сильніший відрив). Власні повідомлення – `bg-primary text-primary-foreground` праворуч, чужі – `bg-white` ліворуч. Часова мітка – `text-[10px]` праворуч і в нижньому кутку кожної бульки. Поле введення – Textarea з `resize-none`, `field-sizing-content` (авторозширення висоти), Enter надсилає (Shift+Enter – новий рядок). Прокрутка до нового повідомлення через `containerRef.scrollTop = containerRef.scrollHeight` (а не `scrollIntoView` – щоб уникнути scroll всієї сторінки на мобільних).

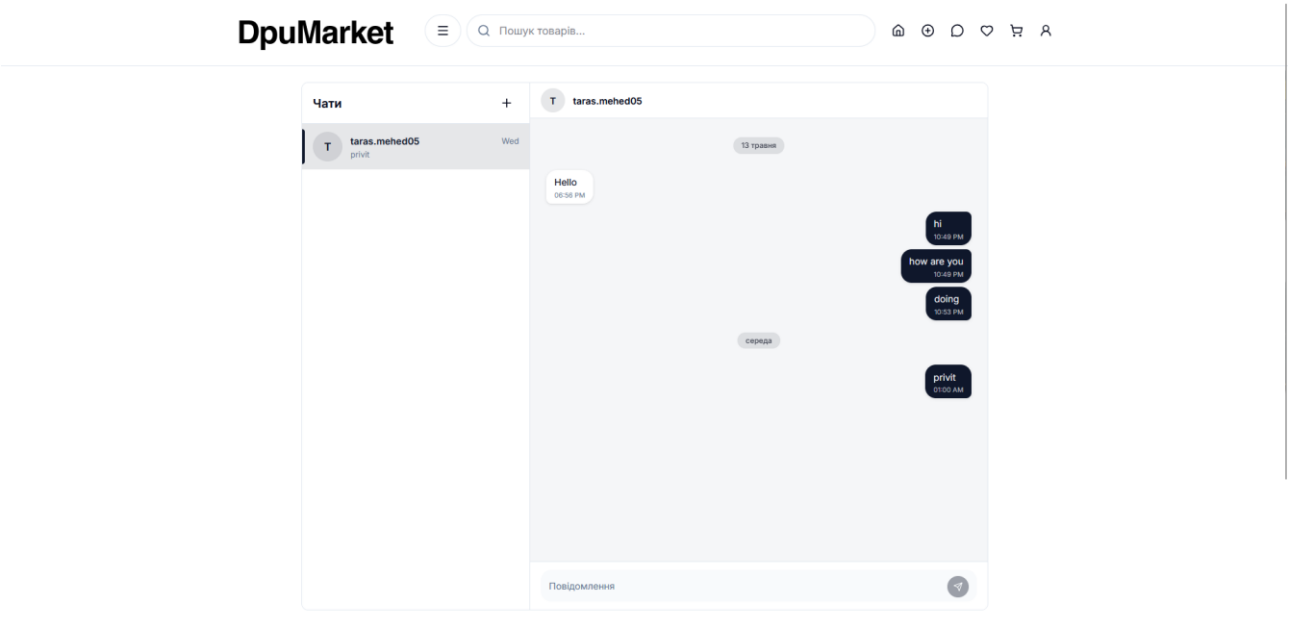


Рис. 2.4. Сторінка чату у стилі Telegram (скриншот реалізації)

Джерело: розроблено автором

Сторінка входу реалізована як єдина сторінка з трьома станами-контекстами (SignIn, SignUp, ConfirmSignUp), перемикання між якими відбувається без перезавантаження через `useState<AuthMode>`. Поле email зберігається між станами, щоб не змушувати користувача вводити його повторно. Кожне поле містить відповідну іконку зліва (Mail, Lock, User) через абсолютне позиціонування. Кнопки валідації і стани завантаження реалізовані через єдиний

`isLoading`-прапорець. Обгортка `<Suspense>` необхідна для `useSearchParams()` при static export – без неї Next.js 15 видає попередження при білді.

Форма додавання товару (`/add-product`) підтримує два режими: створення (default) та редагування (якщо в URL є `?edit=id`). При редагуванні форма завантажує існуючий товар, перевіряє `data.ownerId === user.userId` і заповнює поля. Завантаження зображень – drop-zone або клік на `<input type="file" accept="image/*" hidden>` через `fileInputRef.current?.click()`. Попередній перегляд через `URL.createObjectURL(file)` з очищенням при unmount. При наявності існуючого зображення – відображення через `<S3Image />` з кнопками «Замінити» і «X». Захист від подвійного відправлення – `submittingRef.current` ref (а не state), що не спричиняє перерендеру але надійно блокує re-entrant виклики.

Адаптивність реалізована за mobile-first принципом: стилі за замовчуванням для мобільних, breakpoint-префікси `sm:`, `md:`, `lg:`, `xl:` додають стилі для більших екранів. `overflow-x: clip` на html і body запобігає горизонтальному скролу від стирчачих елементів без створення нового scroll-context (на відміну від `overflow-x: hidden`). `padding-left: env(safe-area-inset-left)` і `padding-right: env(safe-area-inset-right)` на body коректно відображають контент на iPhone з Dynamic Island і notch у landscape-орієнтації. Viewport export з `viewportFit: "cover"` у `src/app/layout.tsx` дозволяє вмісту займати всю область екрана.

Висновки до підрозділу 2.1. Реалізовано повноцінний дизайн маркетплейсу методом design-in-code на базі shadcn/ui (new-york) + Tailwind CSS 4 + Radix UI. Дизайн-система використовує OKLCH-кольори, 4px-сітку відступів, Inter/Geist Mono шрифти. Реалізовано всі ключові UX-патерни: оптимістичні оновлення, pre-fill форм, infinite scroll, mobile-first адаптивність із safe-area support. Евристики Нільсена та стандарти Baymard Institute стали конкретними технічними вимогами, а не лише теоретичними принципами.

2.2. Архітектура та проєктування бази даних веб-сайту маркетплейсу

З огляду на вимоги маркетплейсу до масштабованості, швидкості та мінімальних стартових витрат, прийнято рішення побудувати DpuMarket на серверлес-архітектурі AWS Amplify Gen 2. На відміну від класичної трирівневої моделі (фронтенд – власний API-сервер – власна БД), серверлес-підхід виключає необхідність утримувати і адмініструвати серверні процеси: вся бізнес-логіка описана декларативно у TypeScript-файлах, а провіжинг AWS-ресурсів виконується автоматично при деплої.

Загальна архітектура системи включає такі рівні. Фронтенд (Next.js 15, App Router, React 19) розміщений на AWS Amplify Hosting, статичні файли доставляються через Amazon CloudFront CDN з понад 400 точок присутності. API-шар – AWS AppSync GraphQL: автоматично генерує повну GraphQL-схему на основі декларативних TypeScript-моделей, обслуговує Query, Mutation і Subscription операції, перевіряє авторизаційні правила при кожному запиті без коду розробника. База даних – Amazon DynamoDB (регіон eu-central-1, Франкфурт): повністю керована NoSQL БД з on-demand масштабуванням, реплікацією у трьох зонах доступності, гарантованою латентністю у мілісекунди. Файлове сховище – Amazon S3 з підписаними URL через CloudFront. Автентифікація – Amazon Cognito User Pool. Кастомна логіка – AWS Lambda (дві функції: `search-products` і `index-product`).

Серверлес-архітектура маркетплейсу будується на принципі функціональної декомпозиції: замість одного монолітного серверного застосунку, який обслуговує всі типи запитів, кожна окрема функціональність реалізована як ізольована хмарна послуга з власним життєвим циклом, масштабуванням і політикою безпеки. Такий підхід відповідає принципу єдиної відповідальності (single responsibility) на рівні архітектури: AWS AppSync відповідає виключно за GraphQL

API та real-time підписки, Amazon Cognito — за автентифікацію і керування користувачами, Amazon DynamoDB — за зберігання структурованих даних, Amazon S3 — за зберігання файлів, а AWS Lambda — за виконання кастомної бізнес-логіки.

Розподіл відповідальності між сервісами AWS значно спрощує процес розробки і підтримки маркетплейсу. По-перше, кожен сервіс має чітко визначений API і документацію, що зменшує когнітивне навантаження на розробника при роботі з ним. По-друге, відмова одного сервісу не тягне за собою повну деградацію системи: якщо тимчасово недоступний пошуковий Lambda, основна функціональність каталогу через DynamoDB продовжує працювати. По-третє, кожен сервіс масштабується незалежно відповідно до свого навантаження: під час маркетингової кампанії масштабується читання DynamoDB і CloudFront-CDN, а під час масової реєстрації — Cognito.

Особлива увага у спроєктованій архітектурі приділена принципу «stateless»-обробки: жоден компонент системи не зберігає стан запитів між викликами в оперативній пам'яті. Сесійні дані користувача зберігаються виключно у JWT-токенах Cognito, а між запитами всі необхідні дані вибираються заново з DynamoDB. Це дозволяє AWS Lambda та AppSync горизонтально масштабуватися до тисяч паралельних екземплярів без необхідності реалізовувати складну логіку синхронізації стану між ними. На рис. 2.5 наочно зображено всі компоненти системи, їх взаємодію та шари відповідальності.

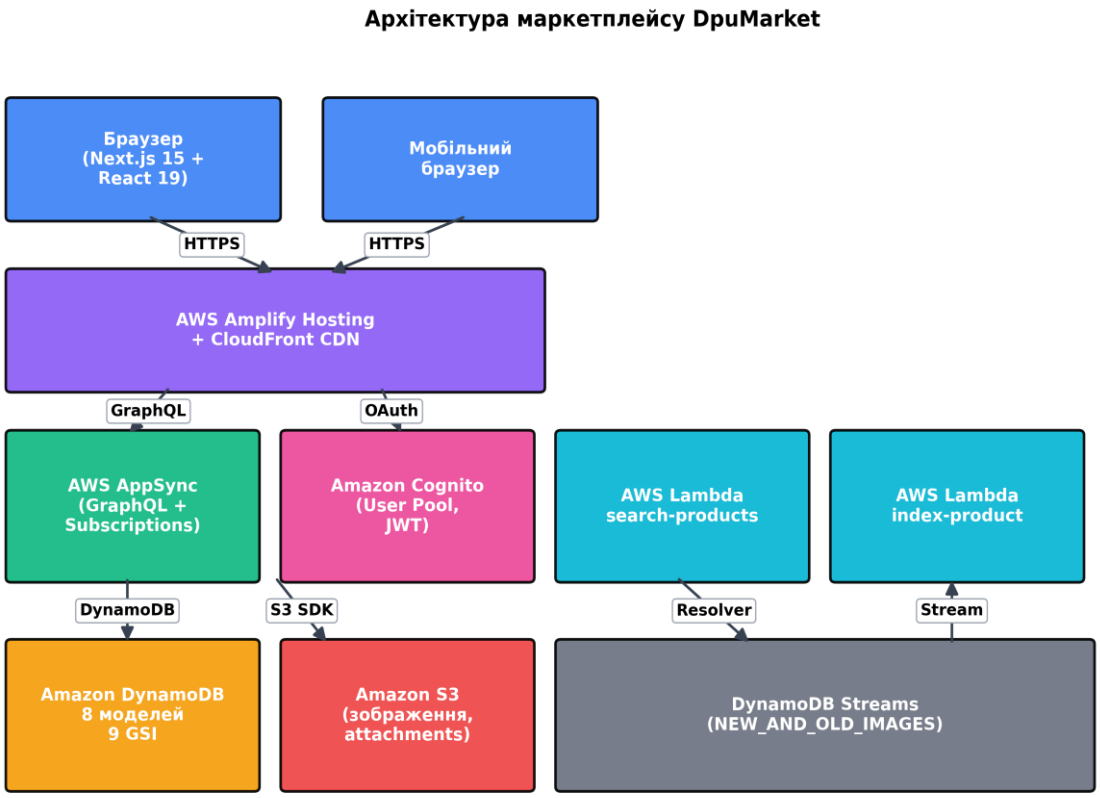


Рис. 2.5. Загальна архітектурна схема маркетплейсу «DpuMarket»

Джерело: розроблено автором

Ключовим файлом Backend є `amplify/backend.ts`. Він підключає auth, data, storage ресурси, а також дві Lambda-функції через CDK escape hatch. Escape hatch – механізм Amplify Gen 2, що дозволяє отримати доступ до нативних CloudFormation/CDK ресурсів і розширити їх будь-якими AWS-примітивами, яких немає у стандартному Amplify API. У `backend.ts` через CDK: надаються права читання DynamoDB-таблиці Product для Lambda `search-products` методом `productTable.grantReadData(searchLambda)`; встановлюється змінна середовища `PRODUCT_TABLE_NAME`; вмикається DynamoDB Streams на таблиці Product (StreamViewType: NEW_AND_OLD_IMAGES); Lambda `index-product` підписується на stream через `DynamoEventSource` з batch size 25, retryAttempts 3, bisectBatchOnError true.

Центральне місце в архітектурі займає модель даних. Маркетплейс DpuMarket реалізує вісім моделей даних у файлі `amplify/data/resource.ts` (повний лістинг наведено в Додатку Б). У таблиці 2.1 наведено характеристики кожної моделі. Повна ER-діаграма бази даних маркетплейсу з усіма зв'язками наведена у додатку А (рис. А.1).

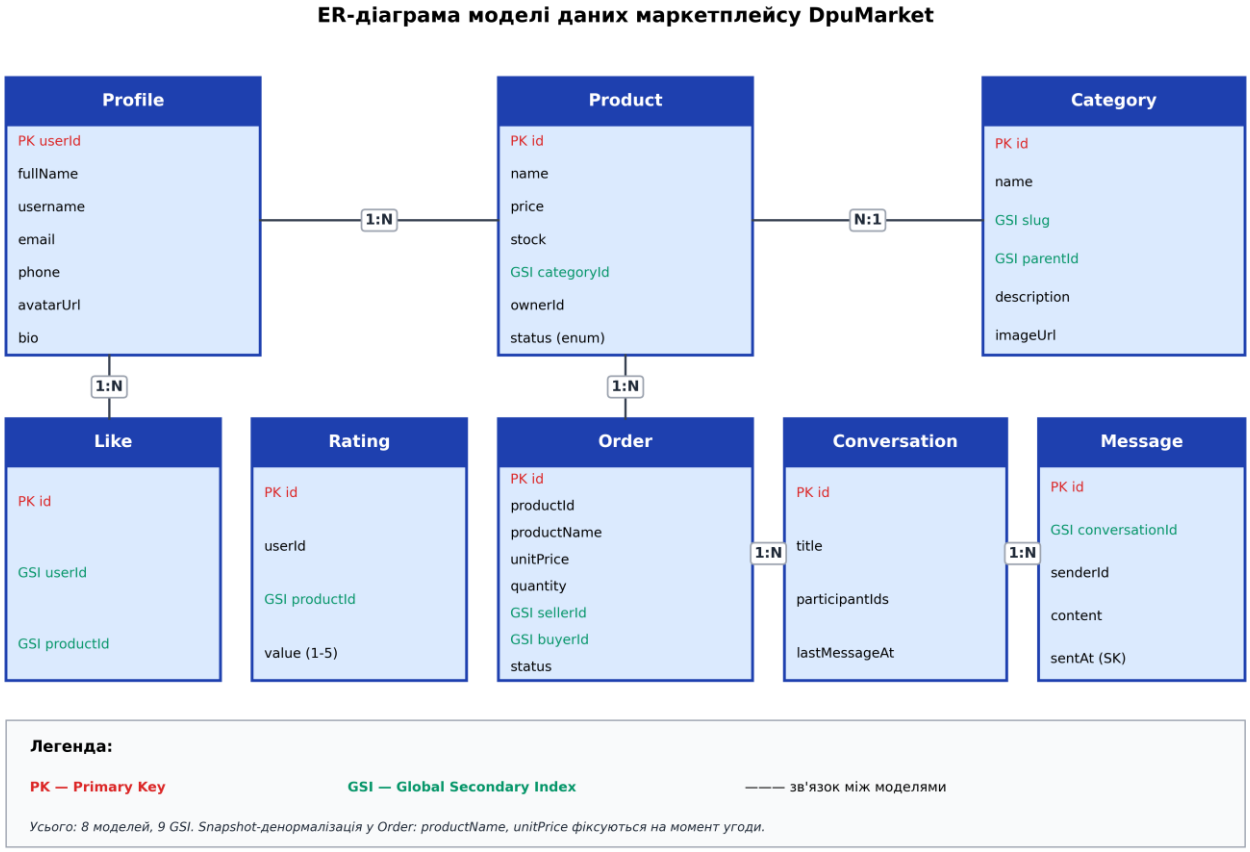


Рис. 2.6. ER-діаграма моделі даних маркетплейсу DpuMarket (8 моделей, 9 GSI)

Джерело: розроблено автором за реальною схемою amplify/data/resource.ts

Таблиця 2.1

Моделі даних маркетплейсу «DpuMarket» з ключовими полями

Модель	Призначення	Ключові поля та особливості
Profile	Профіль Cognito-користувача	userId (custom identifier = Cognito sub), fullName, username, email, phone, bio, avatarUrl
Product	Товар у каталозі	id, name, price, stock, category (string), categoryId (→Category), sku, description, imageUrl, ownerId, status enum
Category	Ієрархічна категорія	id, name, slug (унікальний URL-identifier), parentId (adjacency list), description, imageUrl
Like	Список обраних товарів	id, productId, userId; GSI: likesByUser (pk=userId), likesByProduct (pk=productId)
Rating	Зіркова оцінка 1-5	id, productId, userId, value; GSI: ratingsByProduct (pk=productId)
Conversation	Діалог між учасниками	id, title, participantIds[] (масив), lastMessageAt; allow.authenticated (CRUD)
Message	Повідомлення у діалозі	id, conversationId, senderId, senderName, content, sentAt, readAt; GSI: messagesByConversation (pk=conversationId, sk=sentAt)
Order	Замовлення покупця	snapshot-поля: productName, unitPrice, quantity, totalPrice; sellerId, buyerId; GSI: ordersBySeller, ordersByBuyer; status enum

Джерело: складено автором за реальною схемою amplify/data/resource.ts

Між моделями встановлено такі зв'язки: Profile 1:1 Cognito User (userId = Cognito sub); Profile 1:N Product (ownerId); Profile N:M Product через Like; Profile N:M Product через Rating (обмеження: один запис на пару productId+userId); Product N:1 Category (categoryId); Category N:1 Category ієрархічно (parentId – adjacency list); Conversation 1:N Message; Profile N:M Profile через Conversation (participantIds[]); Product 1:N Order (productId snapshot); Profile 1:N Order як seller (sellerId) і як buyer (buyerId).

Особливістю моделювання у DynamoDB є свідома денормалізація. Модель Order містить snapshot-поля: `productName`, `productImageUrl`, `unitPrice` – дублювання даних із Product. Це є обґрунтованим рішенням: якщо продавець

пізніше змінить назву або ціну товару, інформація про фактичну угоду залишиться незмінною у записі Order. Аналогічно, поле `senderName` у Message дозволяє відобразити стрічку чату одним GSI-запитом без N+1 звернень до Profile для кожного повідомлення – так само, як це реалізовано у Facebook Messenger та Telegram. Денормалізація є не порушенням, а свідомим архітектурним патерном NoSQL, який дає O(1) час відповіді на типові запити[2].

Критично важливим аспектом проєктування є вибір правил авторизації. У DpuMarket застосовано нестандартний підхід `ownerDefinedIn` замість стандартного `allow.owner()`. Причина: при наявності явного поля `ownerId` (або `userId`, `senderId`) Amplify не автоматично заповнює стандартне поле `owner` на create, тому `allow.owner()` порівнює JWT sub з null і відхиляє кожен запит. Рішення – `allow.ownerDefinedIn("ownerId").identityClaim("sub")`: правило явно вказує, яке поле містить ідентифікатор власника і який claim JWT порівнювати. Це задокументована особливість Amplify Gen 2, що вирішується саме через `ownerDefinedIn`.

У таблиці 2.2 наведено повний перелік глобальних вторинних індексів (GSI) проєкту.

Таблиця 2.2

Глобальні вторинні індекси (GSI) маркетплейсу «DpuMarket»

GSI (queryField)	Модель	Partition Key	Sort Key
productsByCategory	Product	categoryId	—
likesByUser	Like	userId	—
likesByProduct	Like	productId	—
ratingsByProduct	Rating	productId	—
messagesByConversation	Message	conversationId	sentAt
ordersBySeller	Order	sellerId	—
ordersByBuyer	Order	buyerId	—
categoriesBySlug	Category	slug	—
categoriesByParent	Category	parentId	—

Джерело: складено автором. Усі GSI показано на ER-діаграмі в Додатку А.

Пошук товарів реалізовано через кастомний AppSync-запит `searchProducts`, підкріплений Lambda-функцією `search-products`. Функція сканує DynamoDB-таблицю Product з DynamoDB Filter Expression по статусу

ACTIVE, після чого виконує in-memory фільтрацію: текстовий запит (підрядкове включення у полях name, description, category, sku через `toLowerCase().includes(lower)`), фільтрація за категорією, ціновим діапазоном (priceMin, priceMax), наявністю (inStockOnly). З повного відфільтрованого масиву будуються фасети: список категорій з кількістю товарів (`categoryCounts`), цінова гістограма кроком $25 (`histBuckets`), статистика ціни (min, max, avg). Результат сортується (RELEVANCE – порядок DynamoDB, PRICE_ASC/PRICE_DESC – `sort`, NEWEST – за `createdAt`) і пагінується через `from`/`size`. Підхід «DynamoDB scan + in-memory» є компромісом між простотою реалізації та функціональністю для prototype-масштабу (до кількох тисяч товарів). DynamoDB Streams вже активовано і Lambda `index-product` підписана – при зростанні каталогу достатньо реалізувати обробник для запису в OpenSearch без зміни архітектури.

Ієрархія категорій реалізована через підхід «adjacency list»: поле `parentId` вказує на батьківський запис, `parentId = null` – коренева категорія. Дерево будується на клієнті функціями `topLevelCategories(cats)` і `childrenOf(cats, parentId)` з `src/lib/categories.ts`. Для навігації по slug використовується GSI `categoriesBySlug` (`slug` – унікальний URL-identifier, наприклад «electronics», «smartphones»). Для отримання товарів категорії – GSI `productsByCategory` з `partition key = categoryId`. Такий підхід дозволяє отримати всі товари категорії одним запитом (O(1) на рівні DynamoDB), а не скануванням всієї таблиці. Заповнення категорій виконано скриптами `scripts/seed-categories.ts` (8 коренів) і `scripts/rename-categories-ua.ts` (перейменування на українську).

Lambda `search-products` налаштована через `amplify/functions/search-products/resource.ts`: `timeoutSeconds: 30`, `memoryMB: 512`, `resourceGroupName: "data"`. Lambda `index-product` – `timeoutSeconds: 60`, `memoryMB: 512` – but наразі є no-op handler: лише логує вхідні записи стріму без реальних дій. Коментар у коді: "Search is now served by the search-products Lambda scanning DynamoDB directly — no OpenSearch required for prototype scale. The stream subscription

is kept so this can be repurposed later." Таке рішення дозволяє включити повноцінний пошуковий рушій в майбутньому без зміни інфраструктури – stream вже активний, Lambda підписана, залишається лише реалізувати обробник.

Файлове сховище Amazon S3 налаштоване через `amplify/storage/resource.ts` з трьома шляхами доступу. `products/*` – зображення товарів: читання гостями через IAM unauthenticated (`allow.guest.to(["read"])`), запис/видалення автентифікованими (`allow.authenticated.to(["read","write","delete"])`). `avatars/{entity_id}/*` – аватари: повний доступ власнику через Cognito Identity ID (`allow.entity("identity").to(["read","write","delete"])`), читання гостями. `chat-attachments/*` – вкладення чату: тільки автентифіковані. Підписані URL генеруються через `getUrl({ path, options: { expiresIn: 3600 }})` з терміном 1 година. Аватари зберігаються за шляхом `avatars/{identityId}/{timestamp}.jpg`, де identityId – Cognito Identity ID (отримується через `fetchAuthSession().identityId`), що відповідає правилу `entity("identity")` у Storage policy.

Система автентифікації побудована через Amazon Cognito User Pool. Конфігурація у `amplify/auth/resource.ts`: вхід через email (`loginWith.email`), верифікація email через одноразовий CODE, кастомний текст листа через verificationEmailBody: (createCode) => template string з кодом, парольна політика: мінімум 8 символів, обов'язково lowercase, uppercase, digits, symbols. Додаткові атрибути `preferredUsername` і `profilePicture` (mutable, required=false). Cognito видає ID Token, Access Token та Refresh Token за OAuth 2.0/OIDC; AppSync автоматично перевіряє токени без коду розробника. AmplifyProvider налаштовує `CookieStorage` із динамічним `secure: isHttps` – на localhost HTTP cookies зберігаються без HTTPS, на production – тільки HTTPS, що вирішує проблему "Unable to get user session following successful sign-in" при роботі в dev-середовищі. Функція `ensureProfileExists` автоматично створює Profile-запис при першому вході нового користувача.

Безпека даних у серверлес-архітектурі забезпечується на кількох рівнях без додаткового коду розробника. Паролі: не зберігаються у DpuMarket –

управляються Cognito (bcrypt + salt). Ін'єкції: AppSync використовує параметризовані DynamoDB-запити (Expression Attribute Values) – SQL-ін'єкції принципово неможливі в NoSQL DynamoDB. XSS: React автоматично екранує всі значення при рендерингу. DDoS: AWS Shield Standard вбудований у CloudFront і AppSync без додаткового налаштування. Шифрування: DynamoDB і S3 шифрують дані at-rest за замовчуванням (AES-256). Transport: HTTPS enforced через Amplify Hosting і CloudFront. Авторизація: кожен AppSync-запит перевіряє правила `allow.ownerDefinedIn`, `allow.authenticated`, `allow.publicApiKey` – несанкціонований доступ блокується на рівні сервісу.

Масштабування системи відбувається автоматично без участі розробника. DynamoDB у режимі on-demand (pay-per-request) автоматично збільшує Read/Write Capacity Units при зростанні навантаження і знижує при спаді. Lambda автоматично запускає до 3000 паралельних виконань на регіон. AppSync горизонтально масштабується як керований сервіс. Все це означає, що при різкому зростанні трафіку у 10-50 разів (типово для e-commerce в дні розпродажів) маркетплейс продовжить функціонувати стабільно без будь-яких операційних дій. Проєкт DpuMarket розгорнуто у регіоні eu-central-1 (Франкфурт) для мінімальної латентності для українських та європейських користувачів.

У таблиці 2.3 наведено основні GraphQL-операції API маркетплейсу.

Таблиця 2.3

Основні GraphQL-операції API маркетплейсу «DpuMarket»

Тип	Операція	Призначення
Query	listProducts (filter: status=ACTIVE)	Каталог товарів з пагінацією (nextToken)
Query	searchProducts	Пошук через Lambda: DynamoDB scan + фасети
Query	productsByCategory (categoryId)	Товари категорії через GSI
Query	likesByUser (userId)	Обрані товари (GSI)
Query	ratingsByProduct (productId)	Оцінки товару (GSI)
Query	messagesByConversation	Повідомлення (GSI, sort: sentAt)
Query	ordersBySeller / ordersByBuyer	Замовлення за роллю (GSI)
Mutation	createProduct / updateProduct	CRUD товарів (ownerDefinedIn)

Mutation	createOrder	Замовлення (auth або publicApiKey)
Subscription	onCreateMessage	Real-time нові повідомлення через WebSocket

Джерело: складено автором. Повний код API наведено в Додатку Б.

Моніторинг і спостережуваність системи базуються на вбудованих AWS-сервісах. Amazon CloudWatch Metrics автоматично збирає показники Lambda-функцій (тривалість виконання, кількість викликів, throttles, errors) та AppSync-запитів (latency p50/p99, 4xx/5xx error rates). CloudWatch Logs агрегує структуровані JSON-логи усіх Lambda і AppSync; у Lambda `search-products` відображається кількість відсканованих товарів і час фільтрації. AWS X-Ray трасує повний шлях запиту від AppSync через Lambda до DynamoDB, виявляючи вузькі місця продуктивності. Amplify Hosting надає дашборд CI/CD: статус кожного деплою, логи збірки, час розгортання, завершені артефакти.

Детальніший розгляд моделі Category показує гнучкість підходу adjacency list. Кожна категорія має slug (унікальний URL-ідентифікатор) і parentId (посилання на батька або null). GSI categoriesBySlug дозволяє за O(1) знайти категорію за URL без сканування. GSI categoriesByParent за O(1) повертає всі підкатегорії. Заповнення відбувається через scripts/seed-categories.ts (8 коренів), переклад на українську через scripts/rename-categories-ua.ts. Прив'язка товарів через поле categoryId у Product і GSI productsByCategory покриває запит за одне звернення.

Модель Order реалізована зі snapshot-денормалізацією: при оформленні замовлення у запис копіюються productName, productImageUrl, unitPrice — актуальні на момент угоди значення, що не змінюються при подальших правках товару продавцем. Це стандартна практика в Amazon, Shopify та Rozetka. Гостьовий checkout (guest checkout) реалізовано через allow.publicApiKey().to(["create"]) для Order — замовлення можна оформити без реєстрації. Для кожного CartItem створюється окремий Order-запис, що дозволяє продавцям різних товарів незалежно бачити свої замовлення через GSI ordersBySeller.



Рис. 2.7. Стан-машина життєвого циклу замовлення Order

Джерело: розроблено автором

CartProvider реалізує клієнтський кошик без синхронізації з DynamoDB — свідоме рішення для підтримки гостьового сценарію. Стан персистується у localStorage (ключ tarasmehed_cart). Гідратація безпечна: початковий стан порожній масив на сервері, localStorage читається у useEffect після mount. LikesProvider реалізує дедуплікацію: при завантаженні likesByUser дублікатні записи автоматично видаляються. Захист від race condition через inFlight Set ref — повторний toggleLike при незавершеному попередньому ігнорується.

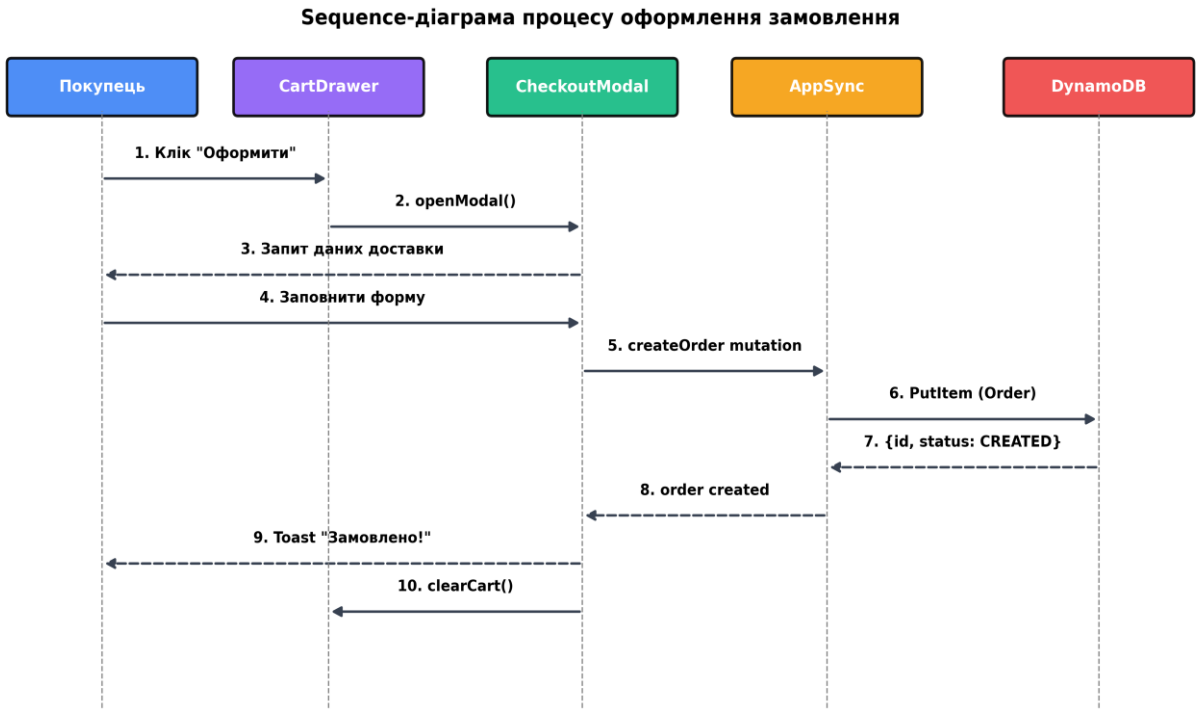


Рис. 2.8. Sequence-діаграма процесу оформлення замовлення (Checkout)

Джерело: розроблено автором

AmplifyProvider виконує три додаткові функції. По-перше, CookieStorage з динамічним secure прапором: на localhost HTTP cookies дозволені, на production — тільки HTTPS. По-друге, ensureProfileExists при кожному signedIn: якщо Profile-запису немає — створюється stub. По-третє, Hub-слухач синхронізує auth-контекст у всіх компонентах при signedIn, signedOut, tokenRefresh. SSR-підтримка через src/middleware.ts: fetchAuthSession оновлює cookie-сесію перед рендерингом Server Component, що дозволяє виконувати authenticated AppSync-запити на сервері.

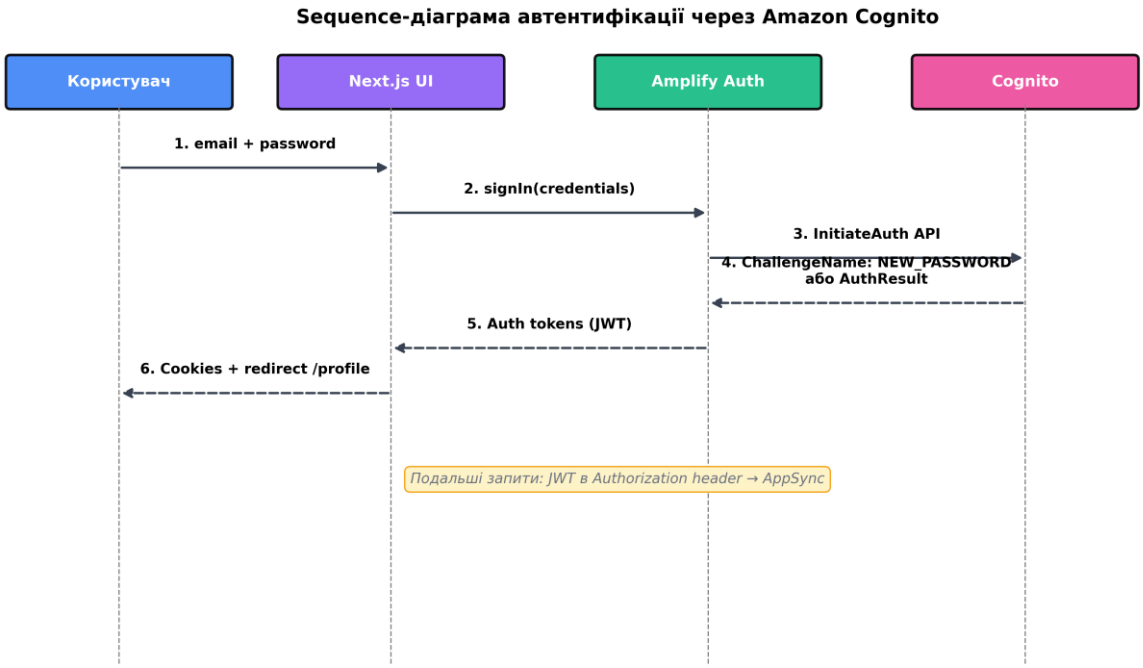


Рис. 2.9. Sequence-діаграма процесу автентифікації користувача через Amazon Cognito

Джерело: розроблено автором

З архітектурної точки зору GraphQL надає переваги перед REST для маркетплейсу. Field selection усуває over-fetching: для сітки карток запитується id, name, price, imageUrl, ownerId; для деталей — усі поля. Real-time підписки AppSync через MQTT over WebSocket надають push-сповіщення чату без polling. Сильна типізація через Schema = ClientSchema<typeof schema> синхронізує схему API і TypeScript-типи клієнта автоматично при кожному деплої. Авторизаційні правила декларовані безпосередньо поряд зі схемою моделі та застосовуються AppSync без коду розробника.

З точки зору безпеки та відповідності вимогам GDPR[13] і PCI DSS: паролі керуються Cognito; банківські дані карток не зберігаються в DpuMarket; DynamoDB і S3 шифрують at-rest через AES-256; CloudFront і AppSync забезпечують HTTPS; AWS Shield Standard захищає від DDoS; Point-in-Time Recovery DynamoDB зберігає backup за 35 днів. Авторизаційні правила ownerDefinedIn гарантують, що кожен запис доступний лише його власнику або авторизованим

читачам — несанкціонований доступ блокується на рівні AppSync без коду на клієнті.

Детальний розгляд сторінки пошуку та каталогу розкриває важливі технічні рішення. Компонент ProductSearch завантажує всі ACTIVE-товари через paginated list calls (limit=100 per page, пагінація через nextToken поки не null) і передає масив до FilterableProductGrid. Такий підхід — «завантажити все, фільтрувати in-memory» — є оптимальним для каталогу до 1000-2000 товарів: одна мережева операція замість N операцій при кожній зміні фільтру. При зростанні каталогу Lambda search-products повністю замінює цей підхід через DynamoDB scan + фасети — архітектурний перехід відбудеться без змін інтерфейсу. FilterableProductGrid реалізує useMemo для обох операцій: мемоізована фільтрація (залежності: products, query, priceMin, priceMax, qtyMin, qtyMax, fixedCategory) і мемоізоване сортування (залежність: filtered, sort). Це забезпечує O(1) перерахунок при відсутності змін у залежностях і уникає зайвих re-render.

Компонент S3Image вирішує нетривіальну задачу відображення зображень з Amazon S3. S3-обєкти не мають публічних URL за замовчуванням; для читання потрібні підписані URL з обмеженим терміном дії. S3Image отримує S3-шлях (наприклад, products/23e428b2.../1748000000000.jpg), викликає getUrl({ path, options: { expiresIn: 3600 }}) і передає отриманий підписаний URL у next/image. Термін 1 година є компромісом між безпекою (URL не діє вічно) і зручністю (сторінка залишається функціональною протягом типового сеансу перегляду). Компонент показує спінер Loader2 під час отримання URL і fallback-вміст при помилці або відсутності шляху. Legacy-URL (починаються з http:// або https://) використовуються напряму без виклику getUrl.

Компонент StarRating реалізує дробові зірки через CSS clip-path. Для кожної з п'яти позицій обчислюється fillPct = Math.max(0, Math.min(1, display - (i-1))) * 100, де display — поточне значення рейтингу. Внутрішня зірка (жовта fill-yellow-400) ширина якої задана через style={{ width: fillPct + "%" }}, накладається поверх сірої зірки-контуру. Це забезпечує будь-які дробові значення: 3.7

зірки відображаються з семи десятих заповненою четвертою зіркою. Інтерактивний режим (prop onChange): hover встановлює integer-значення через setHover, клік зберігає оцінку через onChange(i). При submitRating перевіряється наявність існуючого запису в ratings для userId — якщо є, виконується update замість create, що забезпечує принцип "один голос на користувача".

Структура Provider-ієрархії у кореневому layout.tsx визначає правильний порядок: AmplifyProvider (зовнішній, надає auth-context) обгортає LikesProvider (залежить від auth), який обгортає CartProvider (незалежний від auth, але знаходиться всередині для консистентності), далі Header (потребує CartProvider для badge), main (дочірні сторінки), Footer, CartDrawer (потребує CartProvider). CartDrawer не є дочірнім елементом Header — він рендерується на рівні layout і монтується через Radix Dialog Portal у тіло документа, що уникає проблем z-index і overflow:hidden на проміжних контейнерах.

Механізм реального часу (real-time) у чаті базується на GraphQL Subscriptions AppSync через MQTT over WebSocket. При відкритті сторінки чату встановлюється WebSocket-зєднання із AppSync endpoint. Для активного діалогу — підписка `client.models.Message.onCreate({ filter: { conversationId: { eq: activeConversationId }}})` миттєво доставляє нові повідомлення. Глобальна підписка на всі діалоги оновлює превью останнього повідомлення і переставляє список за lastMessageAt. Дедуплікація через `prev.some(m => m.id === newMessage.id)` запобігає відображенню дублікатів у разі перепідписки. Кожна підписка повертає unsubscribe-функцію, що чиститься у useEffect cleanup — запобігає memory leaks при unmount компонента. Реалізація відповідає шаблону з документації AWS AppSync.

Локалізація і інтернаціоналізація (i18n). Весь інтерфейс DpuMarket реалізований українською мовою. Це включає: всі кнопки (Додати товар, Оформити замовлення, Вийти), заголовки сторінок (Вподобані товари, Мої товари, Отримані замовлення), статуси (Активний, Архівний, Очікує, Підтверджено, Відправлено, Виконано, Скасовано), повідомлення про помилки і успіх, підказки

у формах, назви полів. Ціни відображаються у гривнях (знак ₴) замість $ з попередньої версії. Формат часу — через date-fns у locale uk. Вибір мови є стратегічним: маркетплейс орієнтований на внутрішній ринок України, а україномовний інтерфейс є вимогою Закону України про забезпечення функціонування державної мови в цифровому просторі.

Оптимізація продуктивності реалізована на кількох рівнях. Next.js App Router за замовчуванням серверно рендерить компоненти, що не потребують стейту (Header, Footer, метадані), суттєво зменшуючи JavaScript-бандл клієнта. Turbopack (--turbopack у dev і build) пришвидшує компіляцію у 10-700 разів порівняно з webpack за рахунок Rust-реалізації і послідовної інкрементальної компіляції. Оптимізація зображень через next/image автоматично конвертує у WebP/AVIF, генерує srcset для різних розмірів екранів, додає lazy loading.

Висновки до розділу 2. У другому розділі спроєктовано повноцінний маркетплейс DpuMarket. Реалізовано design-in-code UX/UI-підхід на основі shadcn/ui (new-york) + Tailwind CSS 4 + Radix UI + OKLCH-кольорів. Спроєктовано серверлес-архітектуру AWS Amplify Gen 2 з 8 моделями даних (Profile, Product, Category, Like, Rating, Conversation, Message, Order), 9 GSI, Lambda-пошуком через DynamoDB scan, декларативною авторизацією ownerDefinedIn + identityClaim("sub"), snapshot-денормалізацією у Order, ієрархічними категоріями (adjacency list), S3-сховищем і Cognito з CookieStorage для SSR. Архітектура є готовою основою для практичної реалізації третього розділу.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-САЙТУ МАРКЕТПЛЕЙСУ «DpuMarket»

У третьому розділі представлено результати практичної реалізації веб-сайту маркетплейсу DpuMarket на основі проєктних рішень, обґрунтованих у розділах 1 та 2. Розробка велася у форматі ітеративних спринтів з безперервною інтеграцією через AWS Amplify Hosting. Для управління задачами використовувалася дошка GitHub Projects; для контролю версій – приватний Git-репозиторій на GitHub з класичним git flow (main, develop, feature/*).

Проєкт ініційовано командою `create-next-app` з подальшою інтеграцією AWS Amplify Gen 2 через `npm create amplify@latest`. Структура репозиторію типова для Next.js 15 + Amplify Gen 2: директорія `src/app/` – сторінки та компоненти за File-Based Routing; `src/components/` – повторно використовувані React-компоненти, провайдери контексту та UI-примітиви; `amplify/` – декларативні конфігурації бекенд-ресурсів (`auth/resource.ts`, `data/resource.ts`, `storage/resource.ts`, `backend.ts`) та Lambda-функції; `scripts/` – допоміжні утиліти для seed-даних; `src/lib/` – спільні функції (category helpers, Amplify SSR utilities). Для керування залежностями використано npm з обов'язковим прапором `--legacy-peer-deps` через неповну сумісність React 19 з рядом бібліотек на момент розробки.

Перший робочий білд створено за принципом «walking skeleton»: мінімальна сторінка Next.js та базовий Amplify-бекенд через Amplify Sandbox – локальне хмарне середовище, яке провіжить AWS-ресурси в ізольованому стеку розробника й оновлює їх у реальному часі при зміні конфігурацій. Це дозволило перевірити всю інфраструктуру (автентифікація Cognito, GraphQL AppSync, DynamoDB, S3) ще до написання значного обсягу коду.

Реалізацію функціональності розподілено на такі спринти: декларативний опис моделей та правил авторизації (`amplify/data/resource.ts`, фрагмент якого наведено у додатку Б); автентифікація через Cognito; каталог і детальні сторінки товарів; вподобайки та зіркові оцінки; кошик (CartProvider + CartDrawer + CheckoutModal); замовлення (Order модель); категорії (Category модель + seed-скрипти); пошук (Lambda `search-products`); чат з real-time підписками; профіль з 4 вкладками; оптимізація й деплой.

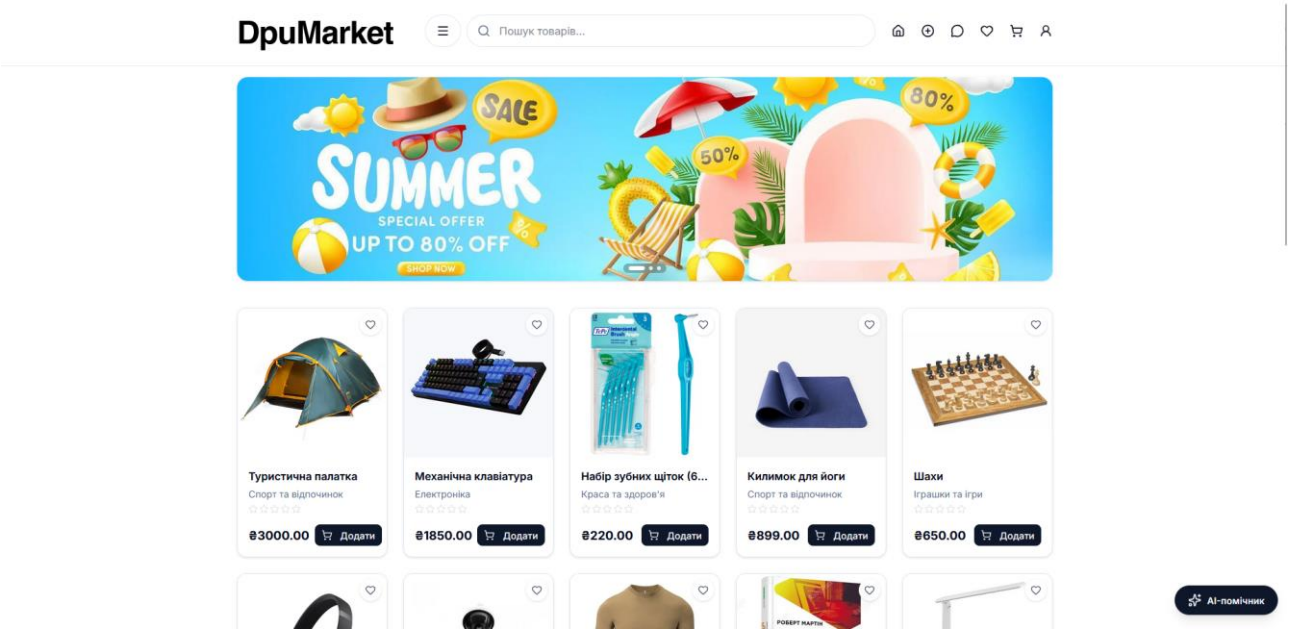


Рис. 3.1. Головна сторінка веб-сайту маркетплейсу «DpuMarket» (скриншот реалізації)

Джерело: розроблено автором

Фронтенд реалізовано на Next.js 15 з App Router та React 19. React Server Components використано для статичних частин (Header, Footer, метадані сторінок), що суттєво зменшує JavaScript-бандл, який надсилається клієнту. Інтерактивні компоненти (кошик, фільтри, модальні вікна, чат) реалізовано з директивою `"use client"`. Збирання виконується через Turbopack – новий високошвидкісний bundler від команди Next.js (прапор `--turbopack` у скриптах dev і build). Для управління глобальним станом використано три власні React Context-провайдери: `AmplifyProvider` (авторизація, типобезпечний Amplify-клієнт), `LikesProvider` (список обраних товарів), `CartProvider` (кошик). Вибір нативних

механізмів React замість Zustand чи Redux обґрунтований тим, що три ізольованих контексти простіші в дебагу та не вносять зовнішніх залежностей.

Стилізація виконана в Tailwind CSS 4 (`@tailwindcss/postcss`) з набором компонентів shadcn/ui. Компоненти не є зовнішньою бібліотекою – вони копіюються у проєкт і адаптуються під дизайн. Радіус заокруглень – 0.625rem. Кольорова система побудована на CSS-змінних у форматі oklch – первинний колір, фони, межі – без жодних hardcoded значень. Це забезпечує тривіальне впровадження темного режиму через клас `dark`. Складна інтерактивна поведінка компонентів (Dialog, Accordion, Slider, Drawer) реалізована через Radix UI primitives, що гарантує відповідність стандарту WAI-ARIA без вбудованих стилів. Іконки – lucide-react; карусель на головній сторінці – embla-carousel-react; шрифти – Inter (основний) і Geist Mono, завантажені через `next/font/google`. Карусель на головній сторінці відображає три рекламні банери (`/banners/summer-one.jpg`, `/banners/summer-two.jpg`, `/banners/summer-three.jpg`) з автопрокруткою кожні 5 секунд.

Реалізовано 10 функціональних сторінок: `/` (головна з banner-каруселлю та нескінченною сіткою товарів), `/product/[id]` (деталі товару з оцінюванням, лайком, кошиком), `/add-product` (додавання і редагування товару – edit mode через query-параметр `?edit=id`), `/products` (управління власними товарами), `/profile` (чотири вкладки: Профіль, Товари, Продажі, Покупки), `/liked` (вподобані товари), `/chat` (месенджер з real-time), `/search` (пошукова сторінка), `/category/[slug]` (каталог категорії), `/login` (єдина сторінка з трьома станами: вхід, реєстрація, підтвердження email).

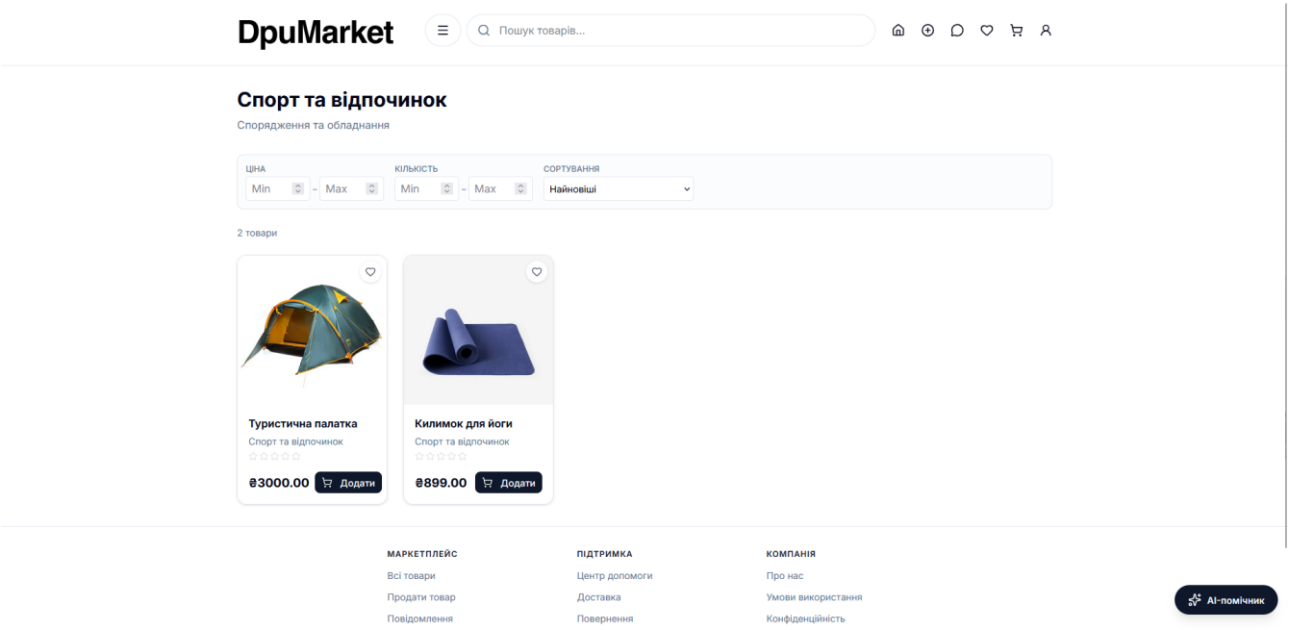


Рис. 3.2. Сторінка каталогу категорії з фільтрами (реалізація)

Джерело: розроблено автором

Бекенд побудовано декларативно через три Amplify Gen 2-файли. Файл `amplify/auth/resource.ts` описує Cognito User Pool: вхід через email, підтвердження одноразовим кодом, кастомний текст листа-підтвердження, парольна політика (мінімум 8 символів, великі й малі літери, цифри, спеціальні символи), додаткові атрибути `preferredUsername` та `profilePicture`. Файл `amplify/data/resource.ts` містить схему 8 моделей з декларативними правилами авторизації та кастомний GraphQL-запит `searchProducts`. Файл `amplify/storage/resource.ts` налаштовує Amazon S3 з трьома шляхами доступу: `products/*` (читання гостями через IAM, запис автентифікованими), `avatars/{entity_id}/*` (повний доступ власнику через Cognito Identity, читання гостями), `chat-attachments/*` (тільки автентифіковані).

Файл `amplify/backend.ts` підключає дві Lambda-функції через CDK escape hatch. Перша – `search-products`: AppSync Lambda resolver для кастомного запиту `searchProducts`, що виконує повний скан DynamoDB-таблиці Product, in-memory фільтрацію та повертає пагінований результат з фасетами. Функція отримує ім'я таблиці через змінну середовища

`PRODUCT_TABLE_NAME`, яку Amplify додає автоматично. Права читання таблиці надаються через `productTable.grantReadData(searchLambda)`. Друга – `index-product`: підписана на DynamoDB Streams таблиці Product (StreamViewType: NEW_AND_OLD_IMAGES), але наразі є no-op placeholder – логує вхідні події без реальних дій. Stream збережено для майбутньої інтеграції з пошуковим рушієм.

Lambda `search-products` реалізує повноцінний пошуковий рушій без OpenSearch: повний скан DynamoDB з фільтром по статусу ACTIVE, потім in-memory фільтрація за текстовим рядком (підрядкове включення у полях name, description, category, sku), категорією, ціновим діапазоном та наявністю на складі. Далі обчислюються фасети: список категорій з кількістю товарів і цінова гістограма в кроках $25. Результат сортується (RELEVANCE, PRICE_ASC, PRICE_DESC, NEWEST) і пагінується через параметри from/size. На фронтенді компонент `ProductSearch` використовує простіший спосіб – пряму вибірку через `client.models.Product.list` з подальшою передачею до `FilterableProductGrid`. Таким чином Lambda готова до підключення при зростанні каталогу, а поточна in-memory логіка покриває весь функціонал для прототипу.

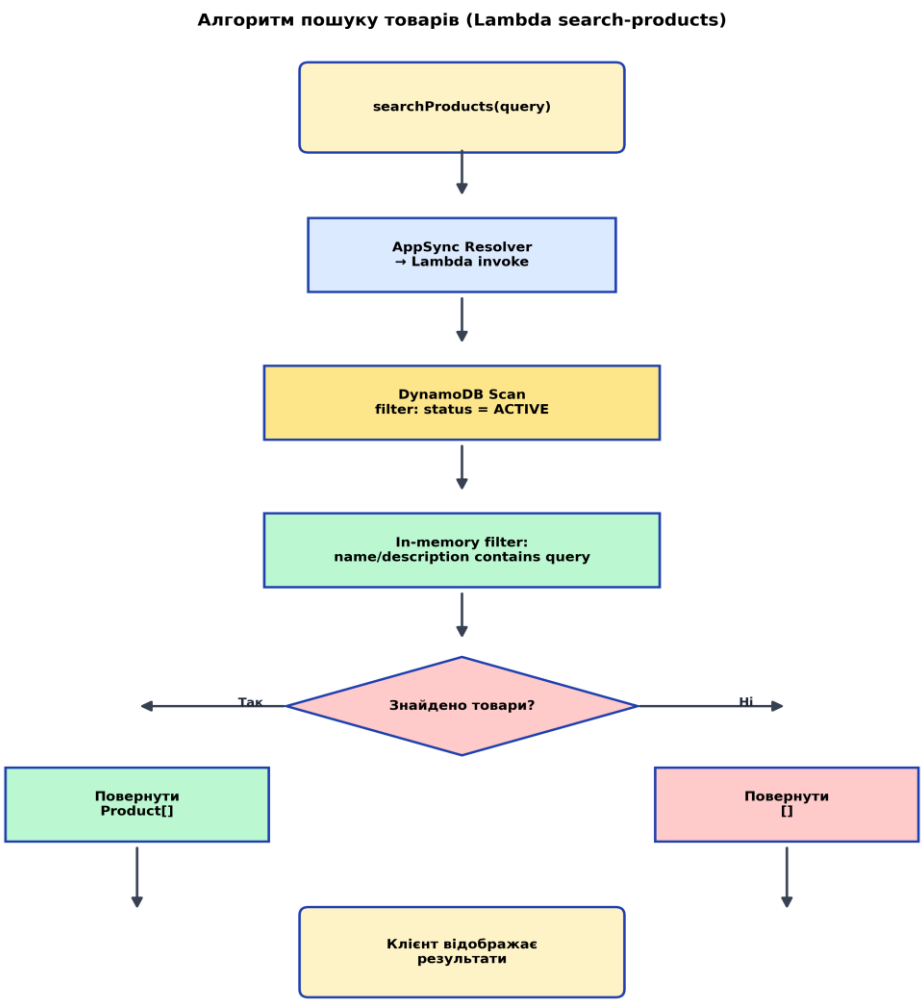


Рис. 3.3. Алгоритм пошуку товарів через Lambda search-products

Джерело: розроблено автором

Автентифікація реалізована власною сторінкою `/login` з трьома станами: SignIn, SignUp, ConfirmSignUp. Для взаємодії з Cognito використано функції з `aws-amplify/auth`. Авторизаційні токени зберігаються через `CookieStorage` з динамічним прапором `secure` залежно від протоколу. SSR-доступ до сесії забезпечує `src/middleware.ts` через `fetchAuthSession`.

Сторінка детальної картки товару (`/product/[id]`) є однією з найважливіших у маркетплейсі — саме тут відвідувач ухвалює рішення про додавання товару в кошик. У реалізації використано Server Component для рендерингу основних даних товару (назва, ціна, опис, залишок на складі), що дозволяє

пошуковим системам індексувати сторінку без необхідності виконання JavaScript. Динамічні елементи — поле кількості, кнопки «Додати до кошика» і «У вподобані», блок оцінювання — реалізовано як клієнтські компоненти ('use client'), оскільки вони потребують доступу до React Context для оновлення глобального стану кошика і списку вподобань.

Метадані сторінки (title, description, Open Graph для соцмереж, JSON-LD структуровані дані за схемою Product) генеруються через експорт `generateMetadata()` Next.js на основі реальних даних товару з DynamoDB. Це забезпечує коректне відображення товару при розшаренні в Telegram, Facebook та інших платформах. Для зображень товару використовується компонент `<Image>` від Next.js з автоматичною оптимізацією, lazy loading при прокручуванні та адаптивним srcSet для різних розмірів екрану.

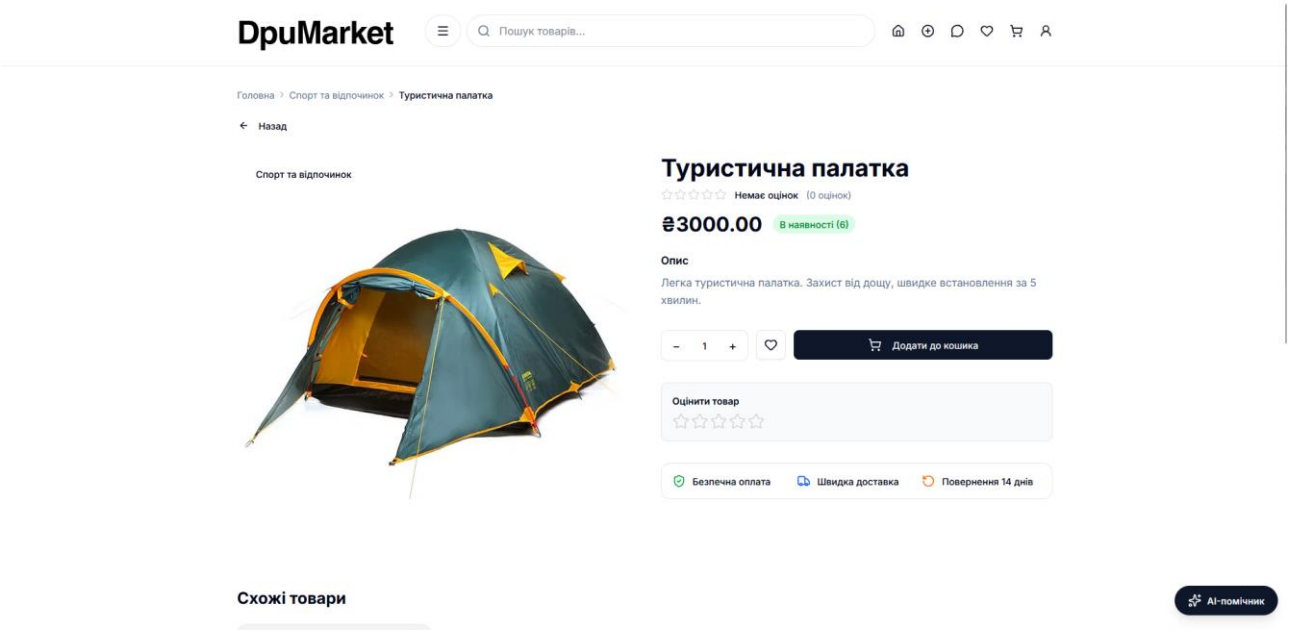


Рис. 3.4. Сторінка детальної картки товару (реалізація)

Джерело: розроблено автором

При першому успішному вході `AmplifyProvider` автоматично викликає функцію `ensureProfileExists`, яка перевіряє наявність запису в моделі Profile і, якщо його немає, створює stub-запис із userId (= Cognito sub), email та автоматично генерованим username (частина email до @). Це гарантує, що кожен

зареєстрований користувач одразу стає видимим у пошуку при створенні нових чатів, навіть якщо він ще не заповнив профіль вручну.

Кошик реалізовано через `CartProvider` з персистентністю у `localStorage` (ключ `tarasmehed_cart`). Гідратація безпечна: початковий стан – порожній масив і на сервері, і на першому клієнтському рендері, а `localStorage` зчитується в `useEffect` після монтування – стандартний патерн уникнення hydration mismatch. `CartDrawer` відкривається з правого боку екрана і показує повний список товарів з можливістю зміни кількості (обмеженої `maxStock`), видалення позицій і суми. При натисканні «Оформити замовлення» відкривається `CheckoutModal`: він попередньо завантажує Profile поточного користувача і заповнює поля fullName, email, phone. Кожен товар у кошику оформляється окремим записом Order (один Order = одна позиція), що спрощує логіку GSI `ordersBySeller`/`ordersByBuyer` при відображенні вкладок Продажі і Покупки у профілі.

Профіль користувача (`/profile`) організовано у чотири вкладки з лічильниками: Профіль (редагування даних і аватара), Товари (список і управління оголошеннями продавця), Продажі (отримані замовлення від покупців – через GSI `ordersBySeller`), Покупки (зроблені замовлення – через GSI `ordersByBuyer`). Вкладки реалізовані без зовнішніх бібліотек: власне CSS з indicator-лінією та sticky-позиціонуванням. Відображення лічильника у бейджі на мобільних версіях реалізоване через абсолютно позиційований span поверх іконки, на десктопі – відкрито поряд з текстом. Управління статусами замовлень для продавця (PENDING → CONFIRMED → SHIPPED → COMPLETED, можливість CANCELLED) реалізовано кнопками безпосередньо у картці замовлення.

Завантаження зображень реалізовано напряму з браузера в Amazon S3 через метод `uploadData()` з Amplify Storage, що дозволяє уникнути передачі файлів через бекенд. Перед завантаженням зображення обробляється на стороні клієнта: через Canvas API картинка масштабується до максимального розміру 1200×1200 пікселів зі збереженням пропорцій, заповнюється білим фоном для прозорих

PNG, та конвертується у JPEG зі стисненням 85% (константа `IMAGE_QUALITY = 0.85`). Це знижує розмір файлу у 3-10 разів без помітної втрати якості. В S3 зображення зберігаються за шляхом `products/{userId}/{timestamp}.jpg` (де userId – Cognito sub). Для аватарів використовується шлях `avatars/{identityId}/{timestamp}.jpg`, де `identityId` – Cognito Identity ID, що відповідає правилу `entity("identity")` у Storage access policy. Для відображення зображень реалізовано компонент `<S3Image />`, який отримує підписаний URL через `getUrl()` з терміном 1 година та передає його в `<Image />` від Next.js.

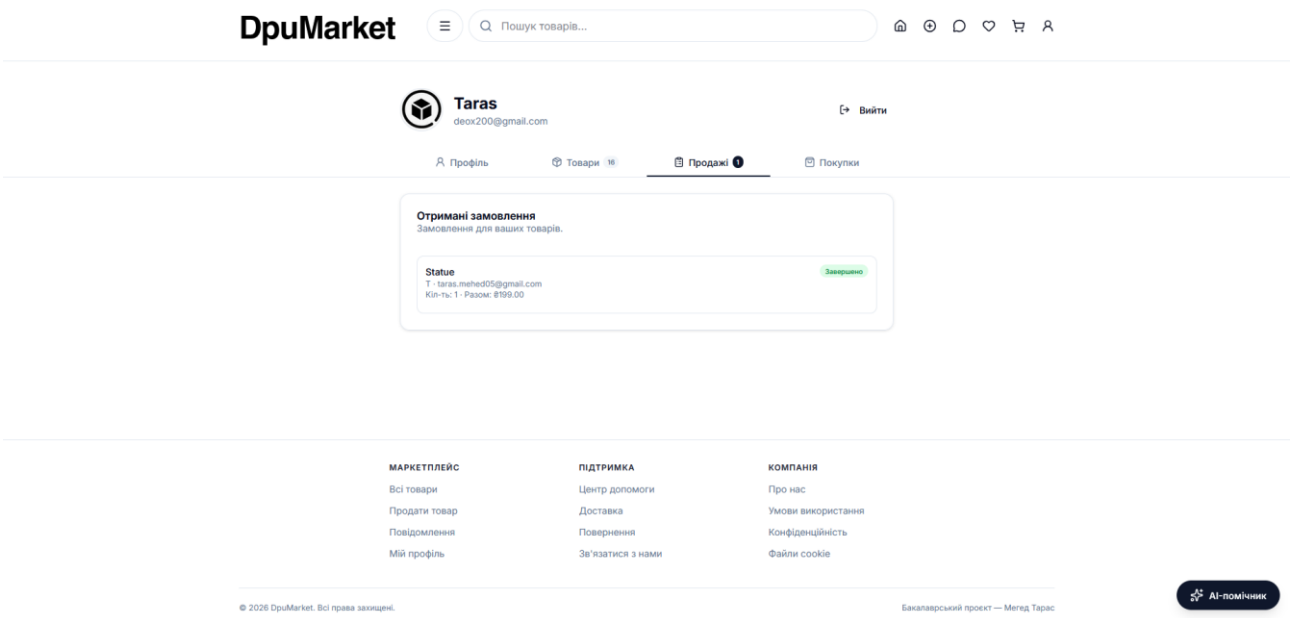


Рис. 3.5. Сторінка профілю з вкладкою «Продажі» (реалізація)

Джерело: розроблено автором

Чат реалізовано у двопанельному компонуванні: список розмов (ліва панель) + активний діалог (права панель). На мобільних пристроях панелі перемикаються. Real-time доставка нових повідомлень забезпечується GraphQL Subscription `client.models.Message.onCreate` – AppSync встановлює WebSocket-з'єднання і миттєво доставляє нові записи без polling. Активний діалог підписується на повідомлення свого `conversationId`, а глобальний обробник підписується на всі діалоги користувача для оновлення прев'ю і сортування списку. Нові повідомлення дедуплікуються за `id` перед додаванням до стейту.

Автопрокрутка реалізована через `containerRef.scrollTop = containerRef.scrollHeight` (а не `scrollIntoView`), що виключає небажаний scroll сторінки на мобільних пристроях. Форма введення надсилає по Enter (Shift+Enter – новий рядок) і автоматично розширюється у висоту. Перелік учасників чату – масив `participantIds[]` у моделі Conversation.

Категорії реалізовано через модель Category з ієрархічною структурою (adjacency list). Для наповнення каталогу розроблено seed-скрипти у `scripts/`: 8 кореневих категорій (Електроніка, Одяг, Дім і сад, Книги, Іграшки, Спорт, Краса, Авто) та українськомовні товари. Навігація між категоріями – через GSI `categoriesBySlug` та `productsByCategory`.

Форма додавання товару (`/add-product`) підтримує режим редагування через query-параметр `?edit=id`. При цьому сторінка завантажує існуючий товар, перевіряє `ownerId === user.userId` (захист від несанкціонованого редагування), заповнює форму і при збереженні викликає `client.models.Product.update`. Submitting guard реалізовано через `useRef<boolean>` (а не через стейт) – це виключає повторні відправки при швидких подвійних кліках, оскільки ref не викликає перерендеру. Після успішного збереження нового товару відбувається редирект на `/product/{id}`, після редагування – на `/profile`.

Авторизаційна схема у моделях Amplify використовує власний підхід `ownerDefinedIn` замість стандартного `allow.owner()`. Проблема: при наявності явного поля `ownerId` Amplify не автоматично заповнює стандартне поле `owner`, тому `allow.owner()` порівнює поточного користувача з null і відхиляє кожен запит. Рішення – `allow.ownerDefinedIn("ownerId").identityClaim("sub")`: правило явно вказує, яке поле моделі містить ідентифікатор власника і який JWT-claim використовувати для порівняння. Аналогічно для Profile: `allow.ownerDefinedIn("userId").identityClaim("sub")`.

Для забезпечення якості коду в проєкт інтегровано статичний аналіз (ESLint 9 у форматі Flat Config `eslint.config.mjs` з плагінами `eslint-plugin-react`, `eslint-plugin-react-hooks`, `@next/eslint-plugin-next`, `typescript-eslint`, `eslint-

config-prettier`), форматування коду (Prettier з плагіном `prettier-plugin-tailwindcss` для автоматичного впорядкування Tailwind-класів), суворий TypeScript у режимі strict (але `noEmit: true`, перевірка через `tsc` окремо від збірки). Важливо: у `next.config.ts` встановлено `eslint: { ignoreDuringBuilds: true }` – ESLint не блокує production-збірку при попередженнях, що відповідає стандартній практиці для прототипів. CI/CD повністю забезпечується AWS Amplify Hosting через файл `amplify.yml`: при кожному пуші в Git виконується `npm install --legacy-peer-deps` та `npm run build`, артефакти з директорії `.next` розгортаються автоматично. Кеш `.next/cache/**` і `node_modules/**` між збірками прискорює deployment.

Підсумковим результатом третього розділу є повнофункціональний прототип маркетплейсу DpuMarket, розгорнутий на AWS Amplify Hosting у регіоні eu-central-1. Реалізовано 10 функціональних сторінок; 8 моделей даних; 9 GSI; 2 Lambda-функції; повний кошик з localStorage; ієрархічні категорії; real-time чат; завантаження зображень у S3.

Економічний ефект від впровадження DpuMarket: для власника – скорочення витрат на запуск порівняно з аутсорсинговою розробкою аналогічного продукту при нульових інфраструктурних витратах на старті завдяки AWS Free Tier; для продавців – розширення ринку збуту; для покупців – зручний пошук і порівняння цін.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання – здійснено розробку веб-дизайну та реалізацію веб-сайту маркетплейсу DpuMarket на сучасному серверлес-стеку. Отримані результати дозволяють зробити такі узагальнюючі висновки.

1. Проведено аналіз ринку електронної комерції: показано, що маркетплейси займають домінуючу позицію у структурі глобальної e-commerce та демонструють стійке зростання в Україні. Виявлено ключові функціональні вимоги до сучасного маркетплейсу – двобічна аудиторія, зручний пошук і каталог, захищені замовлення, комунікація між учасниками, адаптивний мобільний інтерфейс – та проаналізовано їх реалізацію у провідних конкурентів (Rozetka, Prom.ua, OLX, Amazon, Etsy).

2. На основі порівняльного аналізу сучасних технологій обґрунтовано вибір технологічного стеку: фронтенд – Next.js 15 з React 19 та Tailwind CSS 4; бекенд – серверлес-платформа AWS Amplify Gen 2 (AWS AppSync GraphQL, AWS Lambda); база даних – Amazon DynamoDB (NoSQL, регіон eu-central-1); автентифікація – Amazon Cognito; файлове сховище – Amazon S3 з CDN CloudFront. Показано, що саме такий стек забезпечує автоматичне масштабування, глобальну швидкість, нульові стартові витрати (AWS Free Tier), безпеку за замовчуванням та відповідає архітектурним рішенням провідних маркетплейсів (Amazon, Shopify, Airbnb, Etsy).

3. Розроблено UX/UI-дизайн маркетплейсу методом design-in-code – без окремого прототипування у Figma, безпосередньо у процесі розробки. Основу дизайн-системи склали shadcn/ui (стиль "new-york"), Tailwind CSS 4 з CSS-змінними в форматі oklch, Radix UI для accessible-компонентів, Inter + Geist Mono для типографіки. Проєктування спиралося на 10 евристик юзабіліті Я. Нільсена. Весь інтерфейс реалізований українською мовою.

4. Спроєктовано серверлес-архітектуру маркетплейсу на базі AWS Amplify Gen 2: 8 моделей даних (Profile, Product, Category, Like, Rating, Conversation, Message, Order) з денормалізацією під шаблони доступу NoSQL; 9 глобальних вторинних індексів (GSI) для ефективних запитів без сканування; декларативна схема авторизації через `ownerDefinedIn` + `identityClaim("sub")`; кастомний GraphQL-запит `searchProducts` через Lambda-резолвер (DynamoDB scan + in-memory фільтрація + фасети); DynamoDB Streams на таблиці Product з підпискою Lambda `index-product` (no-op placeholder для майбутнього рушія).

5. Здійснено практичну реалізацію повнофункціонального прототипу маркетплейсу DpuMarket у суворому TypeScript. Реалізовано 10 сторінок, кошик з localStorage-персистентністю (CartProvider + CartDrawer + CheckoutModal з pre-fill з Profile), повний цикл замовлення (Order model + GSI ordersBySeller/ordersByBuyer + 4-вкладковий профіль з управлінням статусами), ієрархічні категорії з seed-скриптами, пошук з in-memory фільтрацією, real-time чат через GraphQL Subscriptions, завантаження зображень у S3 з клієнтським resize (Canvas → 1200px, JPEG 85%), SSR-middleware для Cognito, auto-create Profile stub при вході. CI/CD забезпечено платформою AWS Amplify Hosting через `amplify.yml`.

6. Маркетплейс DpuMarket підтверджує, що сучасна серверлес-архітектура на AWS Amplify Gen 2 дозволяє одному розробнику за обмежений час побудувати продукт, порівнянний за функціональністю з повноцінними комерційними платформами. Відсутність власного серверного застосунку, автоматичне масштабування та нульові стартові витрати роблять цей підхід оптимальним для нових маркетплейсів, що виходять на ринок.

Розгорнутий веб-сайт маркетплейсу доступний за посиланням: https://tarasmehed.com.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Маркетплейс DpuMarket. Офіційний веб-сайт. URL: https://tarasmehed.com (дата звернення: 20.05.2026).
2. Маркетплейс DpuMarket. Скриншот головної сторінки інтерфейсу. URL: https://tarasmehed.com (дата звернення: 20.05.2026).
3. Маркетплейс DpuMarket. Скриншот сторінки деталей товару. URL: https://tarasmehed.com/products (дата звернення: 20.05.2026).
4. Маркетплейс DpuMarket. Скриншот сторінки профілю користувача. URL: https://tarasmehed.com/profile (дата звернення: 20.05.2026).
5. Маркетплейс DpuMarket. Скриншот інтерфейсу real-time чату. URL: https://tarasmehed.com/chat (дата звернення: 20.05.2026).
6. Amazon Web Services. AWS Amplify Gen 2 Documentation. URL: https://docs.amplify.aws (дата звернення: 10.03.2026).
7. Amazon Web Services. Amazon DynamoDB Developer Guide. URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ (дата звернення: 12.03.2026).
8. Amazon Web Services. Amazon Cognito Developer Guide. URL: https://docs.aws.amazon.com/cognito/latest/developerguide/ (дата звернення: 12.03.2026).
9. Amazon Web Services. Amazon S3 Developer Guide. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ (дата звернення: 13.03.2026).
10. Amazon Web Services. AWS AppSync Developer Guide. URL: https://docs.aws.amazon.com/appsync/latest/devguide/ (дата звернення: 14.03.2026).
11. Amazon Web Services. AWS Lambda Developer Guide. URL: https://docs.aws.amazon.com/lambda/latest/dg/ (дата звернення: 15.03.2026).

12. Baymard Institute. E-Commerce UX Research. URL: https://baymard.com (дата звернення: 20.03.2026).
13. Cooper A., Reimann R., Cronin D., Noessel C. About Face: The Essentials of Interaction Design. 4th ed. Wiley, 2022. 720 p.
14. Duckett J. JavaScript & jQuery: Interactive Front-End Web Development. Wiley, 2022. 640 p.
15. Forrester Research. The State of Commerce Technology. Report. 2023. 48 p.
16. Frost B. Atomic Design. Ann Arbor: Brad Frost, 2023. 194 p.
17. Gerber E., Carroll M. The Skeptic's Guide to Information Architecture. Rosenfeld Media, 2023. 212 p.
18. General Data Protection Regulation (EU) 2016/679. Official Journal of the European Union. 2023.
19. Google Developers. Core Web Vitals. URL: https://web.dev/vitals/ (дата звернення: 18.03.2026).
20. GraphQL Foundation. GraphQL Specification. URL: https://spec.graphql.org (дата звернення: 10.03.2026).
21. Hartson R., Pyla P. The UX Book: Agile UX Development. Morgan Kaufmann, 2024. 942 p.
22. Internet Engineering Task Force. JSON Web Token (JWT). RFC 7519. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 22.03.2026).
23. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. New Riders, 2022. 216 p.
24. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2023. 432 p.
25. Melton J. E-Commerce Design Patterns. O'Reilly Media, 2021. 328 p.
26. Meta. React 19 Documentation. URL: https://react.dev (дата звернення: 05.03.2026).

27. Mozilla Developer Network. Web technology for developers. URL: https://developer.mozilla.org (дата звернення: 15.03.2026).
28. Newman S. Building Microservices. 2nd ed. O'Reilly Media, 2021. 620 p.
29. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 01.03.2026).
30. Nielsen J., Pernice K. Eyetracking Web Usability. New Riders Press, 2020. 456 p.
31. Norman D. The Design of Everyday Things. Revised ed. Basic Books, 2022. 368 p.
32. Otto M., Thornton J. Bootstrap Documentation. URL: https://getbootstrap.com/docs (дата звернення: 17.03.2026).
33. Sammer C., Wyllie G. GraphQL in Action. Manning Publications, 2021. 456 p.
34. Schwartz L. UX for Lean Startups. O'Reilly Media, 2020. 240 p.
35. Stack Overflow. Developer Survey 2024. URL: https://survey.stackoverflow.co/2024 (дата звернення: 15.04.2026).
36. State of JS 2024. Annual Survey of JavaScript Ecosystem. URL: https://stateofjs.com/en-US/2024 (дата звернення: 10.04.2026).
37. Statcounter. Desktop vs Mobile vs Tablet Market Share. URL: https://gs.statcounter.com (дата звернення: 20.04.2026).
38. Statista. Global E-Commerce Revenue 2025. URL: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ (дата звернення: 15.04.2026).
39. Stewart T. Serverless Architectures on AWS. 2nd ed. Manning Publications, 2022. 408 p.
40. Tailwind CSS. Documentation. URL: https://tailwindcss.com/docs (дата звернення: 06.03.2026).

41. Tilkov S., Vinoski S. Node.js for Modern Web Applications. IEEE Internet Computing. 2022. Vol. 26, No. 4. P. 62–70.
42. Vercel. Next.js 15 Documentation. URL: https://nextjs.org/docs (дата звернення: 05.03.2026).
43. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 20.03.2026).
44. Wong W. Color Index: Over 1100 Color Combinations. HOW Books, 2020. 304 p.
45. Zhang L., Gu A. Design Tokens in Modern Frontend Development. ACM CHI Conference Proceedings. 2023. P. 1–12.
46. shadcn/ui. Component Library Documentation. URL: https://ui.shadcn.com (дата звернення: 08.03.2026).
47. Radix UI. Primitives Documentation. URL: https://www.radix-ui.com/primitives (дата звернення: 08.03.2026).

ДОДАТКИ

Додаток А

ER-діаграма бази даних маркетплейсу «DpuMarket» (повний варіант)

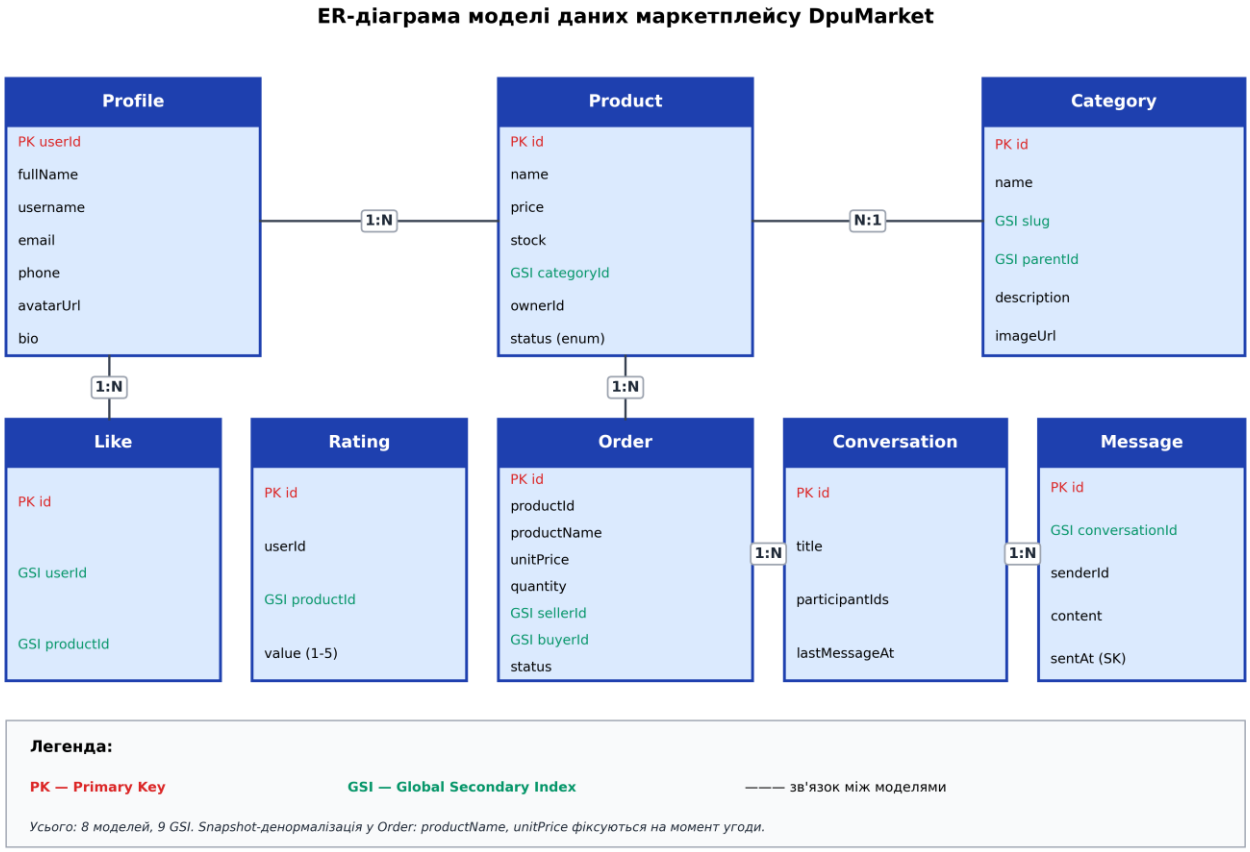


Рис. А.1. ER-діаграма бази даних маркетплейсу «DpuMarket»

На діаграмі відображено всі 8 моделей даних поточної версії (Profile, Product, Category, Like, Rating, Conversation, Message, Order) з повним переліком полів, типами даних та зв'язками. Відображено 9 глобальних вторинних індексів (GSI): productsByCategory, likesByUser, likesByProduct, ratingsByProduct, messagesByConversation, ordersBySeller, ordersByBuyer, categoriesBySlug, categoriesByParent. Діаграму розроблено на основі реальної TypeScript-схеми Amplify Data (amplify/data/resource.ts), що автоматично провіжить відповідні DynamoDB-таблиці та GSI у регіоні eu-central-1. Робоча версія веб-сайту маркетплейсу: https://tarasmehed.com.

Додаток Б

Фрагмент схеми даних маркетплейсу (amplify/data/resource.ts)

Робоча версія маркетплейсу, побудованого на наведеній схемі даних, доступна за посиланням: https://tarasmehed.com.

```
import { type ClientSchema, a, defineData } from "@aws-amplify/backend";
import { searchProductsFn } from "../functions/search-products/resource";

const schema = a.schema({
  Profile: a
    .model({
      userId: a.id().required(),
      fullName: a.string(),
      username: a.string(),
      email: a.email(),
      phone: a.phone(),
      bio: a.string(),
      avatarUrl: a.string(),
      createdAt: a.datetime(),
      updatedAt: a.datetime(),
    })
    .identifier(["userId"])
    .authorization((allow) => [
      allow.ownerDefinedIn("userId").identityClaim("sub")
        .to(["read", "create", "update", "delete"]),
      allow.authenticated().to(["read"]),
    ]),

  Product: a
    .model({
      name: a.string().required(),
      price: a.float().required(),
      stock: a.integer().default(0),
      category: a.string(),
      categoryId: a.id(),
      sku: a.string(),
      description: a.string(),
      imageUrl: a.string(),
      ownerId: a.id().required(),
      status: a.enum(["DRAFT", "ACTIVE", "SOLD", "ARCHIVED"]),
```

```
  })
  .secondaryIndexes((index) => [
    index("categoryId").queryField("productsByCategory"),
  ])
  .authorization((allow) => [
    allow.ownerDefinedIn("ownerId").identityClaim("sub")
      .to(["read", "create", "update", "delete"]),
    allow.authenticated().to(["read"]),
    allow.publicApiKey().to(["read"]),
  ]),

Category: a
  .model({
    name: a.string().required(),
    slug: a.string().required(),
    parentId: a.id(),
    description: a.string(),
    imageUrl: a.string(),
  })
  .secondaryIndexes((index) => [
    index("slug").queryField("categoriesBySlug"),
    index("parentId").queryField("categoriesByParent"),
  ])
  .authorization((allow) => [
    allow.authenticated().to(["read", "create", "update"]),
    allow.publicApiKey().to(["read", "create", "update", "delete"]),
  ]),

Order: a
  .model({
    productId: a.id().required(),
    productName: a.string().required(),
    productImageUrl: a.string(),
    unitPrice: a.float().required(),
    quantity: a.integer().required(),
    totalPrice: a.float().required(),
    sellerId: a.id().required(),
    buyerId: a.id(),
    buyerName: a.string().required(),
    buyerEmail: a.email().required(),
    buyerPhone: a.phone(),
    buyerAddress: a.string().required(),
    notes: a.string(),
    status: a.enum(["PENDING","CONFIRMED","SHIPPED","COMPLETED","CANCELLED"]),
  })
```

```
    .secondaryIndexes((index) => [
      index("sellerId").queryField("ordersBySeller"),
      index("buyerId").queryField("ordersByBuyer"),
    ])
    .authorization((allow) => [
      allow.authenticated().to(["read", "create", "update"]),
      allow.publicApiKey().to(["create"]),
    ]),

  searchProducts: a
    .query()
    .arguments({
      query: a.string(),
      category: a.string(),
      priceMin: a.float(),
      priceMax: a.float(),
      inStockOnly: a.boolean(),
      status: a.string(),
      sort: a.string(),
      from: a.integer(),
      size: a.integer(),
    })
    .returns(a.ref("ProductSearchResult"))
    .handler(a.handler.function(searchProductsFn))
    .authorization((allow) => [
      allow.authenticated(),
      allow.publicApiKey()
    ]),
});

export type Schema = ClientSchema<typeof schema>;
export const data = defineData({
  schema,
  authorizationModes: {
    defaultAuthorizationMode: "userPool",
    apiKeyAuthorizationMode: { expiresInDays: 30 },
  },
});
```