МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

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

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Імплементація сучасних тенденцій розробки ПЗ для е-комерції»

Здобувача вищої освіти другого (магістерського)
рівня,
II курсу, групи КМІ-24-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»

_____ Лащенка Антона Олеговича
(підпис) (прізвище, ім'я, по батькові)

Науковий керівник _____ Погорєловська І. Д., к.е.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми _____ Омельчук Антон Анатолійович, к.т.н
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «02» січня 2026 №10

Завідувач кафедри _____ Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(підпис) (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2026

Реєстрація № 18 21.01.2026 (дата) (підпис відповідальної особи)

Результати перевірки до захисту (до захисту, на доопрацювання) 02.01.2026 (дата) (підпис)

Результат захисту 5 (відмінно) (оцінка) 09.02.2026 (дата) (підпис)

Голова комісії: (підпис) О.С. Дюбанов

Члени комісії: (підпис) А.Ю. Горбовий

(підпис) М.М. Філоненко

(підпис) І.Д. Погорєловська

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 кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		09.02.2026 Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	від 0 до 10	9		
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	47		

Оцінювання кваліфікаційної роботи науковим керівником - 48 балів.

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

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
95	A	5	09.02.2026	Дюбанов О.С. Горбовий А.Ю. Філоненко М.М. Погорєловська І.Д. 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 в економіці»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«11» грудня 2024

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Лащенку Антону Олеговичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: «Імплементація сучасних тенденцій розробки ПЗ для е-комерції»
Затверджена наказом ректора від «10» грудня 2024 № 1852
2. Строк здачі закінченої роботи «02» січня 2026
3. Мета роботи: дослідження ключових сучасних тенденцій у розробці ПЗ для е-комерції та їх практична імплементація у створення сучасного веб-застосунку.
Об'єкт дослідження: процес розробки програмного забезпечення для платформ електронної комерції.
Предмет дослідження: сучасні тенденції, технології та архітектурні патерни розробки конкурентоспроможних е-комерційних рішень.
Орієнтовна база дослідження: комплекс сучасних технологій та практик розробки програмного забезпечення для платформ електронної комерції.
4. Вихідні дані до роботи: системи електронної комерції
5. Перелік графічного матеріалу: структура проєкту, схеми алгоритмів, таблиці порівнянь, копії екранів, що підтверджуть розробку, презентація виступу.
6. Зміст кваліфікаційної роботи: Розділ 1. Огляд сучасних тенденцій розробки програмного забезпечення для електронної комерції. 1.1. Поняття, цілі та класифікація систем електронної комерції. 1.2. Сучасні тенденції розробки ПЗ для е-комерції. Розділ 2. Аналіз інструментів та підходів до створення е-комерційних рішень. 2.1. Огляд популярних платформ і фреймворків для e-commerce. 2.2. Аналіз архітектурних підходів до розробки ПЗ для е-комерції. 2.3. Використання CI/CD та контейнеризації у процесі розробки. Розділ 3. Практична реалізація е-комерційного застосунку. 3.1. Проєктування архітектури та структура системи. 3.2. Зміст процесу розробки функціоналу е-комерційного застосунку. 3.3. Тестування, розгортання та оцінка ефективності проєкту.

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2024	До 15.12.2024
2	**І розділ** (розкрита сутність і стан дослідження наукової проблематики роботи, після чого наведені її загальні характеристики)	30.06.2025	30.06.2025
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	02.09.2025	02.09.2025
4	**ІІІ розділ** (проведені теоретичні і експериментальні дослідження та проектні рішення)	25.11.2025	25.11.2025
5	**Висновки**	13.12.2025	03.12.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	20.12.2025	20.12.2025
7	Отримання відгуку керівника	20.01.2026	20.01.2026
8	Отримання зовнішньої рецензії	23.01.2026	23.01.2026

8. Дата видачі завдання «11» грудня 2024

9. Керівник кваліфікаційної роботи _____ Погорєловська І
(підпис) (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ Омельчук А.А.
(підпис) (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ Лащенко А.О.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Лащенко А.О. Імплементація сучасних тенденцій розробки ПЗ для е-комерції.

Мета роботи: дослідження ключових сучасних тенденцій у розробці ПЗ для е-комерції та їх практична імплементація у створення прототипу сучасного веб-застосунку.

Завдання, які були виконані для досягнення визначеної мети:

- проаналізовано теоретичні основи функціонування систем електронної комерції;
- визначено ключові сучасні тенденції, що формують ринок розробки ПЗ для е-комерції;
- виконано порівняльний аналіз домінуючих архітектурних підходів;
- обґрунтовано вибір технологій та архітектурного підходу для реалізації практичного проєкту;
- розроблено прототип е-комерційного застосунку spacia.ua, що імплементує обрані тенденції.

Об'єкт дослідження: процес розробки програмного забезпечення для платформ електронної комерції.

Предмет дослідження: сучасні тенденції, технології та архітектурні патерни розробки конкурентоспроможних е-комерційних рішень.

Інформаційна база дослідження: комплекс сучасних технологій та практик розробки програмного забезпечення для платформ електронної комерції.

Науково-практична значимість одержаних результатів: удосконалення архітектурного підходу до розробки е-комерційних систем шляхом синергії технологій Next.js та Serverless архітектури, а також у подальшому розвитку методів імплементації WebXR, для чого було розроблено алгоритм асинхронної ініціалізації AR-сесій, що вирішує проблему продуктивності на мобільних пристроях. Практичне значення одержаних результатів визначається створенням повнофункціонального веб-застосунку, який підтверджує ефективність запропонованих архітектурних рішень і може слугувати референсною моделлю для розробки масштабованих PWA-магазинів із підтримкою доповненої реальності.

Дипломна робота містить: 64 сторінки, 2 таблиці, 10 рисунків, список використаних джерел з 46 найменувань, 3 додатки.

Ключові слова: е-комерція, веб-розробка, оптимізація, ефективність, бізнес-процеси, функціональність, мікросервісна архітектура.

Рік виконання кваліфікаційної роботи: 2025-2026

Рік захисту роботи: 2026

ANNOTATION

Lashchenko A.O. Implementation of Modern Trends in Software Development for E-commerce.

The purpose of the thesis: research of key modern trends in software development for e-commerce and their practical implementation in creating a prototype of a modern web application.

Tasks performed to achieve the defined goal:

- theoretical foundations of e-commerce systems functioning were analyzed;
- key modern trends shaping the e-commerce software development market were identified;
- a comparative analysis of dominant architectural approaches was performed;
- the choice of technologies and architectural approach for the implementation of the practical project was substantiated;
- a prototype of the e-commerce application *spacia.ua*, implementing the chosen trends, was developed.

Object of research: the process of software development for e-commerce platforms.

Subject of research: modern trends, technologies, and architectural patterns of developing competitive e-commerce solutions.

Research base: Department of Computer and Information Technologies and Systems of the State Tax University.

Scientific and practical significance of the obtained results: The **scientific significance** of the work lies in improving the architectural approach to developing e-commerce systems through the synergy of Next.js technologies and Serverless architecture, as well as in the further development of WebXR implementation methods, for which an algorithm for asynchronous initialization of AR sessions was developed, solving the performance problem on mobile devices. The **practical significance** of the obtained results is determined by the creation of a fully functional web application, which confirms the effectiveness of the proposed architectural solutions and can serve as a reference model for developing scalable PWA stores with augmented reality support.

The master's thesis contains: 64 pages, 2 tables, 10 figures, a list of references with 46 titles, and 3 appendices.

Keywords: website, web development, optimization, efficiency, business processes, functionality.

Year of qualification work completion: 2025-2026

Year of thesis defense: 2026

ЗМІСТ

ВСТУП **3**

РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ ТЕНДЕНЦІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ **6**

1.1. Поняття, цілі та класифікація систем електронної комерції 6

1.2. Сучасні тенденції розробки ПЗ для е-комерції 9

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ ТА ПІДХОДІВ ДО СТВОРЕННЯ Е-КОМЕРЦІЙНИХ РІШЕНЬ **14**

2.1. Огляд популярних платформ і фреймворків для e-commerce 14

2.2. Аналіз архітектурних підходів до розробки ПЗ для е-комерції 18

2.3. Використання CI/CD та контейнеризації у процесі розробки 21

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ Е-КОМЕРЦІЙНОГО ЗАСТОСУНКУ **24**

3.1. Проєктування архітектури та структура системи 24

3.2. Зміст процесу розробки функціоналу е-комерційного застосунку 30

3.3. Тестування, розгортання та оцінка ефективності проєкту 35

ВИСНОВКИ **40**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **42**

ДОДАТКИ **48**

ВСТУП

Електронна комерція впродовж останнього десятиліття швидко перетворюється в один із фундаментальних стовпів сучасної глобальної економіки. Для України, особливо в контексті прискореної цифровізації, ринок е-комерції стрімко зростає, що, породжує високу конкуренцію. В таких умовах успіх бізнесу вже не визначається лише більшим вибором товарів чи ціною - вирішальну роль відіграють швидкість та функціональність програмного забезпечення, що лежить в основі інтернет-магазину. Користувацький досвід став новою валютою, і компанії, які не встигають за технологічними інноваціями, в результаті втрачають позиції.

Актуальність даної роботи полягає у необхідності систематизації та практичного аналізу сучасних тенденцій у розробці програмного забезпечення для е-комерції. Традиційні монолітні платформи, хоч і досі використовуються, все частіше поступаються місцем гнучким, швидким та високотехнологічним рішенням. Такі тенденції, як мобільна орієнтованість (Mobile-First), впровадження штучного інтелекту (AI) для персоналізації, інтеграція засобів доповненої реальності (AR), а також перехід на хмарні та мікросервісні архітектури, перестали бути футуристичними концепціями і стали обов'язковими вимогами ринку.

Розробники програмного забезпечення сьогодні стикаються з викликом: як ефективно поєднати ці різнорідні технології в єдину, стабільну та масштабовану систему. Це вимагає не лише знання конкретних фреймворків, але й глибокого розуміння сучасних архітектурних підходів. Питання проєктування архітектури веб-систем та електронної комерції широко висвітлені у науковій літературі, проте більшість існуючих досліджень фокусуються або на маркетингових аспектах, або на окремих технічних рішеннях ізольовано одне від одного. Хоча технології прогресивних веб-застосунків (PWA) та доповненої реальності (WebXR) активно розвиваються світовими IT-гігантами, зокрема Google (ініціатива Project Fugu для PWA), Apple (ARKit) та консорціумом Khronos Group (стандарт WebXR), наразі спостерігається брак комплексних досліджень,

присвячених їхній інтеграції в єдину сервіс-орієнтовану архітектуру. Особливої уваги потребує проблема оптимізації продуктивності (Web Vitals) при поєднанні «важкого» AR-контенту з принципами Serverless-обчислень [1], що і зумовило вибір напряму даного дослідження.

Мета кваліфікаційної роботи: дослідження ключових сучасних тенденцій у розробці ПЗ для е-комерції та їх практична імплементація у створення прототипу сучасного веб-застосунку.

Для досягнення мети були визначені наступні *завдання:*

- проаналізувати теоретичні основи функціонування систем електронної комерції;
- визначити ключові сучасні тенденції, що формують ринок розробки ПЗ для е-комерції;
- виконати порівняльний аналіз домінуючих архітектурних підходів;
- обґрунтувати вибір технологій та архітектурного підходу для реалізації практичного проєкту.
- розробити прототип е-комерційного застосунку spacia.ua, що імплементує обрані тенденції.

Об'єкт дослідження: процес розробки програмного забезпечення для платформ електронної комерції.

Предмет дослідження: сучасні тенденції, технології та архітектурні патерни розробки конкурентоспроможних е-комерційних рішень.

Методи дослідження, що були використані в роботі: системний аналіз - при роботі з інформаційними ресурсами за темою дослідження та визначення ключових тенденцій; порівняльний аналіз при оцінюванні переваг та недоліків технологій і архітектур; проєктування - структури веб-застосунку; практичне експериментальне впровадження та прототипування програмного рішення.

Інформаційна база дослідження: комплекс сучасних технологій та практик розробки програмного забезпечення для платформ електронної комерції.

Науково-практична значимість одержаних результатів: Наукова значимість роботи полягає в удосконаленні архітектурного підходу до розробки

е-комерційних систем шляхом синергії технологій Next.js та Serverless архітектури, а також у подальшому розвитку методів імплементації WebXR, для чого було розроблено алгоритм асинхронної ініціалізації AR-сесій, що вирішує проблему продуктивності на мобільних пристроях. Практичне значення одержаних результатів визначається створенням повнофункціонального веб-застосунку, який підтверджує ефективність запропонованих архітектурних рішень і може слугувати референсною моделлю для розробки масштабованих PWA-магазинів із підтримкою доповненої реальності.

Апробація результатів роботи: результати досліджень доповідалися на пленарному засіданні Міжнародної науково-практичної конференції здобувачів вищої освіти та молодих учених «Економіка і бізнес: нові реалії, технологічні тренди, цифрові інновації» (Львів-Ірпінь, 15 травня 2025 р.) [2].

РОЗДІЛ 1. ОГЛЯД СУЧАСНИХ ТЕНДЕНЦІЙ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ

1.1. Поняття, цілі та класифікація систем електронної комерції

Електронна комерція (e-commerce) є широким поняттям, що охоплює будь-яку комерційну транзакцію, проведену за допомогою електронних мереж, насамперед Інтернету. В основі кожної такої транзакції лежить система електронної комерції (СЕК) - комплекс програмного та апаратного забезпечення, що забезпечує взаємодію між продавцем та покупцем, обробку замовлень та проведення платежів.

Ключова мета сучасної е-комерційної системи - це не просто продаж товарів, а створення безперебійного та позитивного клієнтського досвіду (Customer Experience), що веде до підвищення конверсії. У технічному аспекті це досягається через забезпечення високої швидкодії інтерфейсу, адаптивності під мобільні пристрої та впровадження інтерактивних технологій.

У науковій літературі підходи до визначення електронної комерції варіюються від суто технологічних до економічних. Зокрема, у фундаментальних працях К. Лаудона е-комерція розглядається як використання Web-технологій для здійснення бізнес-транзакцій [3]. Водночас, цей технологічний базис створює нові умови для маркетингу: як зазначають В. М. Бондаренко та О. В. Омельяненко, цифрові інструменти відкривають безпрецедентні можливості для точної взаємодії з аудиторією, забезпечуючи глибоку персоналізацію комунікацій [16].

Функціонування системи електронної комерції базується на синхронізації двох основних потоків, які підтримуються програмним забезпеченням:

1. Інформаційний потік: двосторонній обмін даними між клієнтом та сервером (відображення каталогу, характеристик товарів, фільтрація, статус замовлення). Це основа, яка вимагає від системи оптимізації рендерингу та мінімізації затримок (Latency).

2. Фінансовий потік: безпечна обробка транзакцій оплати. Цей аспект вимагає інтеграції з платіжними провайдерами (наприклад, Stripe) та дотримання протоколів захисту даних [1].

З точки зору системної архітектури, теоретична модель функціонування СЕК найчастіше реалізується через трирівневу модель (Three-Tier Architecture), яка забезпечує логічне розмежування процесів [17]:

- Рівень представлення (Presentation Layer): Інтерфейс користувача (Frontend), через який відбувається взаємодія покупця з системою. В сучасних умовах цей рівень часто реалізується як Single Page Application (SPA) або Progressive Web App (PWA) для покращення UX.
- Рівень бізнес-логіки (Application Layer): Серверна частина (Backend), яка обробляє запити, керує кошиком замовлень та сесіями користувачів.
- Рівень даних (Data Layer): Структури даних, що забезпечують зберігання інформації про товари та контент.

Класифікація систем електронної комерції є багатовимірною. За моделлю взаємодії учасників традиційно виділяють:

- B2C (Business-to-Consumer): модель, де компанія продає товари безпосередньо кінцевим споживачам. Саме ця модель є об'єктом дослідження та розробки в даній роботі (інтернет-магазин меблів).
- B2B (Business-to-Business): торгівля між компаніями (наприклад, гуртові платформи).
- C2C (Consumer-to-Consumer): це бізнес-модель, за якої один учасник ринку може виступати у ролі як продавця так і покупця, третя сторона — платформа (маркетплейси), що надає інструменти для торгівлі, часто беручи комісію.

Результати порівняльного аналізу бізнес-моделей наведені у табл. 1.1.

Таблиця 1.1

Критерій класифікації	Тип системи	Характеристика	Приклади реалізації
За учасниками взаємодії	B2C (Business-to-Consumer)	Продаж товарів та послуг кінцевому споживачу.	Rozetka, Amazon, spacia.ua
	B2B (Business-to-Business)	Торговельні операції між компаніями.	Alibaba, Prom.ua
	C2C (Consumer-to-Consumer)	Платформи для взаємодії між приватними особами.	OLX, eBay
За технологічною платформою	SaaS (Software as a Service)	Оренда готового хмарного рішення з обмеженим доступом до коду.	Shopify, Wix
	Open Source CMS	Системи з відкритим кодом, що розгортаються на сервері.	Magento, WooCommerce, OpenCart
	Headless / Custom	Індивідуальна розробка з відокремленим фронтендом.	Netflix, Uber
За типом доступу	Web-based	Класичний доступ через браузер.	Більшість інтернет-магазинів
	Mobile First / PWA	Орієнтовані на мобільні пристрої та офлайн-доступ.	Starbucks PWA, Twitter Lite

Класифікація систем електронної комерції за ключовими ознаками
**сформовано автором на основі джерела* [3].

Окрім моделі взаємодії, критично важливою для інженерного проєктування, є класифікація за технологічним підходом до реалізації:

1. SaaS-рішення (Software-as-a-Service): хмарні платформи за підпискою (наприклад, Shopify), які надають готовий функціонал. Вони забезпечують швидкий запуск, але суттєво обмежені у глибокій кастомізації коду.

2. Платформи з відкритим кодом (Open-Source / CMS): системи, які розгортаються на власному сервері (наприклад, Magento, WooCommerce). Вони дають контроль, але часто мають надлишкову архітектурну складність, що впливає на продуктивність.
3. Кастомна розробка (Custom Development / Headless архітектура): створення системи з використанням сучасних фреймворків (React, Next.js) [4]. Саме цей підхід, реалізований у даній роботі, надає максимальну гнучкість для імплементації унікальних функцій, таких як доповнена реальність (AR), та забезпечення високих показників Web Vitals.

1.2. Сучасні тенденції розробки ПЗ для е-комерції

Електронна комерція сьогодні є динамічним та висококонкурентним середовищем, де успіх бізнесу безпосередньо залежить від якості та функціональності програмного забезпечення. Технологічні інновації та еволюція споживчих очікувань постійно формують нові вимоги до розробки ПЗ для онлайн-торгівлі.

Здійснений аналіз сучасного стану розвитку міжнародної електронної комерції свідчить, що сучасна розробка ПЗ для е-комерції характеризується переходом до мобільної орієнтованості [5].

Зі зростанням кількості користувачів, які здійснюють покупки через смартфони та планшети, першочерговим завданням стає створення зручних та швидких мобільних інтерфейсів [6]. Це включає не лише адаптивний дизайн веб сайтів, що гнучко підлаштовується під різні розміри екранів, але й розробку нативних мобільних застосунків, які забезпечують більш інтегрований та персоналізований досвід.

Крім того, все більшої популярності набувають Progressive Web Apps (PWA), що поєднують переваги вебсайтів та мобільних застосунків [7].

Важливу роль відіграє персоналізація, яка стає ключовим фактором залучення та утримання клієнтів. Сучасні е-комерційні платформи активно

використовують штучний інтелект (AI) для аналізу даних про поведінку, вподобання та історію покупок користувачів [8].

Це дозволяє надавати індивідуалізовані рекомендації товарів, персоналізувати маркетингові кампанії та оптимізувати ціноутворення [9]. Чат-боти на основі AI забезпечують цілодобову підтримку клієнтів, відповідаючи на поширені запитання та допомагаючи з оформленням замовлень.

У контексті зростаючих кіберзагроз та посилення регуляторних вимог, критично важливою тенденцією стає архітектура безпеки за принципом мінімізації даних. Сучасні платформи електронної комерції все частіше відмовляються від зберігання чутливих платіжних даних користувачів на власних серверах, делегуючи цю функцію сертифікованим платіжним шлюзам (через API). Це дозволяє значно знизити ризики витоку інформації. З нормативно-правової точки зору, для українського сегмента ринку обов'язковою умовою є відповідність Закону України «Про захист персональних даних», а також врахування європейських норм GDPR (General Data Protection Regulation) [10].

Дотримання цих стандартів забезпечується не лише юридичними заходами, а й технічними засобами: використанням протоколів HTTPS/TLS та налаштуванням політик безпеки контенту (CSP), що є стандартом для сучасних SPA та PWA-застосунків.

Для забезпечення гнучкості та масштабованості сучасні е-комерційні платформи все частіше використовують мікросервісну архітектуру. Замість монолітного застосунку, система розбивається на невеликі незалежні сервіси, кожен з яких відповідає за певну функціональність (наприклад, управління каталогом, обробка платежів, логістика). Це дозволяє незалежно розробляти, тестувати та розгортати окремі компоненти, що значно спрощує підтримку та оновлення системи. Інтеграція з різними зовнішніми сервісами, такими як CRM, ERP, платіжні шлюзи та логістичні провайдери, здійснюється через API, забезпечуючи безперебійну роботу всього екосистеми е-комерції [11].

Перехід до хмарних технологій також є значущою тенденцією. Хмарні платформи пропонують високу масштабованість, надійність та економічну ефективність, що є критично важливим для обробки великих обсягів даних та пікових навантажень, характерних для онлайн-торгівлі. Використання PaaS та SaaS рішень дозволяє компаніям зосередитися на розробці бізнес-логіки, не витрачаючи ресурси на управління інфраструктурою [12].

Зростання популярності голосових помічників та доповненої реальності (англ. augmented reality, AR) і віртуальної реальності (англ. virtual reality, VR) відкриває нові можливості для взаємодії з покупцями. Оптимізація ПЗ для голосового пошуку дозволяє користувачам швидко знаходити потрібні товари, а технології AR можуть використовуватися для візуалізації товарів у реальному середовищі покупця перед здійсненням покупки (наприклад, віртуальна примірка одягу або розміщення меблів у кімнаті) [13]. VR може створити імерсивний досвід онлайн-шопінгу, наближаючи його до відвідування фізичного магазину.

Логічним продовженням цих інновацій став перехід до концепції Headless Commerce (або «безголової» комерції). Цей архітектурний підхід передбачає повне відокремлення фронтенду (зовнішнього інтерфейсу, який бачить користувач) від бекенду (внутрішньої логіки та бази даних). Взаємодія між цими рівнями відбувається через API. Для розробника це означає можливість вільно змінювати дизайн або впроваджувати нові візуальні рішення, не ризикуючи порушити роботу критично важливих процесів, таких як обробка платежів чи складський облік. Окрім того, такий підхід значно спрощує масштабування бізнесу, дозволяючи легко інтегрувати функціонал продажів у будь-які пристрої — від смарт-годинників та мобільних застосунків до елементів інтернету речей (IoT) [14].

Варто зазначити, що сучасні тенденції розробки ПЗ для е-комерції відображають прагнення до створення більш зручного, персоналізованого, безпечного та інтегрованого досвіду для онлайн-покупців. Мобільна орієнтованість, активне використання штучного інтелекту, забезпечення

надійного захисту даних, перехід до гнучких архітектур та хмарних технологій є ключовими напрямами розвитку. Впровадження інноваційних технологій, таких як AR/VR та голосовий пошук, відкриває нові горизонти для взаємодії з клієнтами. Успіх у сучасній е-комерції вимагає від компаній постійного впровадження новітніх технологій та адаптації до мінливих потреб споживачів [15].

Успішна реалізація вищезгаданих технологічних трендів напряму залежить від обраної архітектурної моделі. Визначення структури системи є не просто технічним завданням, а базовим стратегічним рішенням, яке визначає можливості масштабування платформи, її гнучкість та майбутні витрати на підтримку проєкту. Хоча протягом тривалого часу в індустрії переважав монолітний підхід, сучасні вимоги до швидкості та адаптивності ринку змушують розробників переходити до більш сучасних архітектурних рішень.

Класична монолітна модель передбачає об'єднання всього функціоналу застосунку — від візуального інтерфейсу та бізнес-логіки до механізмів роботи з базою даних — у межах одного цілісного програмного блоку. Традиційними прикладами такої побудови системи виступають ранні версії платформи Magento або вебзастосунки, створені за допомогою фреймворків Django та Ruby on Rails. На старті розробки такий підхід має певні переваги, зокрема наявність єдиної кодової бази та простіший процес тестування.

Проте з розвитком системи стають помітними критичні недоліки. Це проявляється у технологічній інертності, коли, наприклад, впровадження складних AR-модулів у стару архітектуру вимагає надмірних ресурсів. Крім того, виникає ризик каскадних збоїв, а також неможливість вибіркового масштабування, що змушує збільшувати потужність всієї системи навіть при навантаженні лише на окремий елемент, як-от кошик замовлень [17].

Еволюційною відповіддю на ці обмеження стала мікросервісна архітектура. Вона пропонує розділити систему на низку автономних служб, кожна з яких відповідає за конкретну функціональну область (наприклад, окремі сервіси для каталогу, оплати чи профілів користувачів). Комунікація між цими

частинами здійснюється через API-протоколи. Така структура дозволяє оновлювати окремі компоненти (наприклад, сервіс кошика) без зупинки всієї платформи та використовувати різні технологічні стеки для вирішення конкретних завдань. Разом з тим, варто враховувати, що мікросервіси значно ускладнюють управління інфраструктурою, вимагають високого рівня автоматизації (DevOps) та створюють виклики, пов'язані з мережевими затримками й підтримкою актуальності даних у розподіленій системі [18].

Подальшим етапом розвитку хмарних технологій стала безсерверна (Serverless) архітектура, яка найчастіше реалізується через модель «функція як сервіс» (FaaS). У межах цієї парадигми розробник фокусується виключно на написанні бізнес-логіки у вигляді окремих функцій, тоді як питання управління серверами та їх масштабування повністю бере на себе хмарний провайдер, як-от AWS Lambda. Це рішення ідеально підходить для сценаріїв із нестабільним трафіком або для виконання фонових операцій, наприклад, обробки зображень. Головними перевагами тут є висока гнучкість та модель оплати за фактично використаний час виконання коду. Проте необхідно зважати на явище «холодного старту» (затримку при першому запуску функції) та ризик певної залежності від екосистеми обраного хмарного вендора [33].

Таким чином, аналіз сучасного стану галузі засвідчує, що конкурентоспроможність програмного забезпечення для е-комерції досягається шляхом інтеграції інноваційних користувацьких інтерфейсів (PWA, AR) із гнучкими архітектурними моделями (Headless, Serverless), які здатні забезпечити необхідний рівень продуктивності, безпеки та масштабованості системи.

РОЗДІЛ 2. АНАЛІЗ ІНСТРУМЕНТІВ ТА ПІДХОДІВ ДО СТВОРЕННЯ Е-КОМЕРЦІЙНИХ РІШЕНЬ

2.1. Огляд популярних платформ і фреймворків для e-commerce

Вибір технологічного стеку - це технічний етап, що визначає життєздатність е-комерційної платформи. В умовах насиченого ринку, який пропонує безліч готових інструментів, розуміння нюансів кожного підходу стає ключем до балансу між бізнес-амбіціями та технічними обмеженнями.

Сегмент готових рішень ("turnkey solutions") можна умовно поділити на хмарні SaaS-продукти та системи з відкритим кодом (Open Source). Беззаперечним лідером першої категорії є Shopify, що охоплює понад мільйон активних магазинів [20]. Працюючи за моделлю Software-as-a-Service, ця платформа бере на себе весь тягар інфраструктурного менеджменту, пропонуючи натомість інтуїтивний інтерфейс, що не вимагає глибоких технічних компетенцій.

Головна особливість Shopify - це швидкість виходу на ринок (Time-to-Market), адже базовий магазин розгортається за лічені години, одразу маючи інтеграції з платіжними компонентами та логістичними операторами [21]. Втім, недоліком таких систем є жорсткість архітектури. Шаблони, які пропонують такі сервіси, дозволяють лише поверхневу кастомізацію, тоді як серйозні зміни бізнес-логіки майже неможливі або вимагають великих вкладень. Також варто враховувати, що комісійна модель монетизації у таких сервісів може стати фінансовим тягарем при масштабуванні оборотів в майбутньому.

Альтернативою виступає Magento (нині Adobe Commerce) - потужна Open Source система, орієнтована на Enterprise-сегмент. Вона надає повний контроль над архітектурою та ідеально підходить для складних сценаріїв, таких як мультимагазинні конфігурації чи специфічні B2B-процеси [22]. Проте платою за гнучкість є високий поріг входу: розгортання та підтримка Magento вимагають значних серверних потужностей та кваліфікованої команди, а процес оновлення часто перетворюється на дороговартісний виклик через проблеми сумісності.

У ніші малого та середнього бізнесу домінує WooCommerce - плагін, що трансформує WordPress у повноцінний магазин. Його основна перевага - це низький поріг входу та використання знайомої екосистеми WP [23]. Однак, так як це лише надбудова до WordPress, WooCommerce успадковує всі її архітектурні недоліки: при зростанні каталогу продуктивність стрімко падає, а безпека системи стає під загрозою та залежить від регулярності оновлень десятків встановлених плагінів.

Паралельно зростає популярність кастомної розробки на основі архітектури "headless", де клієнтська частина (фронтенд) створюється за допомогою JavaScript-фреймворків. Зокрема, бібліотека React від компанії Meta змінила підходи до побудови користувацьких інтерфейсів завдяки впровадженню компонентної моделі та технології Virtual DOM. Висока швидкість роботи забезпечується алгоритмом Reconciliation (узгодження), який порівнює стан віртуального та реального дерева об'єктів перед кожним оновленням сторінки [24, 25]. Використання однонаправленого потоку даних та механізму хуків (Hooks) зробили структуру додатків більш передбачуваною та зручною для тестування.

Ефективним інструментом для створення рішень професійного рівня став фреймворк Next.js [26]. Дане рішення дозволяє подолати характерну для односторінкових додатків (SPA) проблему з індексацією в пошукових системах за допомогою гібридного рендерингу. Це надає розробнику можливість обирати оптимальну стратегію відображення для кожної сторінки: статичну генерацію (SSG) для незмінного контенту, серверний рендеринг (SSR) для динамічних даних або інкрементальне статичне оновлення (ISR). Остання функція дозволяє актуалізувати інформацію, наприклад, у картках товарів, у фоновому режимі без необхідності повного перезапуску процесу збірки проєкту. Окрім цього, інтегровані засоби оптимізації зображень та система файлового роутингу значно спрощують загальну архітектуру застосунку.

З боку серверної розробки (бекенд) провідну роль займає середовище виконання Node.js [27]. Перехід до концепції "Isomorphic JS", що передбачає

використання однієї мови програмування на різних рівнях системи, суттєво полегшує взаємодію в процесі розробки. Завдяки неблокуючій моделі введення-виведення та механізму Event Loop, Node.js демонструє високу ефективність при одночасній обробці великої кількості з'єднань, що є критично важливим для стабільної роботи платформ електронної комерції. Розвитком цієї концепції в Next.js стали API Routes, які дозволяють реалізовувати бекенд-ендпоінти в межах одного репозиторію. Це наближає розробку до безсерверної (serverless) парадигми, де кожен окремий маршрут функціонує як незалежна хмарна функція [28].

Для реалізації практичного проєкту spacia.ua було обрано саме шлях кастомної розробки на базі стеку React та Next.js, та з використанням хмарної платформи Vercel.

Vercel - це хмарна платформа для хостингу веб-застосунків, яка спеціалізується на роботі з фреймворком Next.js. Її головною перевагою є використання глобальної мережі доставки контенту (Edge Network), що дозволяє завантажувати сайт максимально швидко для користувача з будь-якої точки світу [36]. Дане рішення обумовлене комплексом факторів:

По-перше, архітектурна незалежність дозволяє впроваджувати специфічний функціонал, зокрема інтеграцію засобів доповненої реальності (AR), без використання надлишкових технічних надбудов.

По-друге, такий тип розробки забезпечує високу продуктивність системи: методи розділення коду (code splitting) та оптимізації внутрішніх ресурсів у Next.js дозволяють досягти високих значень показників Core Web Vitals, що часто є проблематичним для громіздких систем управління контентом (CMS).

По-третє, важливим є освітній аспект, оскільки створення системи "з нуля" сприяє глибшому розумінню принципів та патернів проєктування.

Нарешті, використання хмарної платформи автоматизує частину процесів розгортання та підтримки, гарантуючи стабільність високого рівня та уникаючи прив'язки до конкретного постачальника послуг (vendor lock-in), що притаманно закритим комерційним рішенням.

Для уникнення суб'єктивності при остаточному виборі технологічного стеку, замість простого порівняння характеристик, було застосовано метод зваженої матриці рішень (Weighted Decision Matrix). Цей підхід дозволяє кількісно оцінити відповідність кожного рішення специфічним вимогам дипломного проєкту, де критичними факторами є не лише швидкість запуску, а й можливості інтеграції WebXR та PWA. Система оцінювання базується на п'яти критеріях з різними ваговими коефіцієнтами, які відображають їх пріоритетність для досягнення мети роботи (табл. 2.1). Оцінка створювалась за шкалою від 1 до 10.

Таблиця 2.1.

Матриця порівняльного аналізу платформ для розробки е-комерційних систем *

Критерій	Вага	Shopify (SaaS)	Magento (CMS)	Next.js (Custom)
AR / WebXR	0.30	4	6	10
Швидкодія	0.25	7	4	9
MVP Розробка	0.20	10	5	7
Вартість	0.15	5	6	10
PWA / Offline	0.10	3	5	10
Результат	**1.00**	**6.00**	**5.20**	**9.15**

**сформовано автором на основі джерела* [29].

Обґрунтування виставлених балів для ключових критеріїв:

- Гнучкість інтеграції WebXR/AR (Вага: 0.30). Рішення на Next.js отримало максимальні 10 балів, оскільки забезпечує повний контроль над DOM-деревом, що є критично необхідним для нативної інтеграції веб-компонента <model-viewer> та обробки подій AR-сесії. Платформи типу Shopify (4 бали) обмежують доступ до ядра, змушуючи використовувати сторонні плагіни, які часто не підтримують необхідну логіку перемикання режимів.
- Продуктивність та Core Web Vitals (Вага: 0.25). Використання статичної генерації (SSG) у Next.js (9 балів) дозволяє досягти показників LCP (Largest Contentful Paint, час завантаження головного/найоб’ємнішого об’єкту) < 1.0с. У

той час як Magento (4 бали) без складного налаштування серверного кешування демонструє низьку швидкодію через архітектурну вагу.

- Вартість та підтримка (Вага: 0.15). Розгортання на Vercel (10 балів) є безкоштовним для дослідницьких проєктів, тоді як Shopify та Magento вимагають щомісячних витрат на підписку або оренду VPS-сервера, що знижує їх привабливість для стартапу на ранній стадії.

Узагальнюючи результати порівняльного аналізу, можна зробити висновок, що попри вищий поріг входження та триваліший час розробки, кастомний підхід на базі Next.js є найбільш виправданим стратегічним вибором для даного проєкту. На відміну від шаблонних платформ, він забезпечує повну технологічну незалежність, що є критичним для безперешкодного впровадження специфічного функціоналу (зокрема, AR-модулів) та досягнення високих показників швидкодії, які безпосередньо впливають на конкурентоспроможність цифрового продукту.

2.2. Аналіз архітектурних підходів до розробки ПЗ для е-комерції

Архітектурний патерн визначається стратегічним рішенням, від якого безпосередньо залежить економічна ефективність проєкту, його здатність до масштабування та можливість інтеграції новітніх технологій. Універсального рішення в сучасній інженерній практиці не існує, тому проєктування системи базується на багатофакторному аналізі компромісів між гнучкістю розробки, складністю підтримки, вимогами до продуктивності та вартістю інфраструктури. Визначення архітектури, зокрема, виходить за межі технічної площини, перетворюючись на фундамент для забезпечення конкурентоспроможності бізнесу в умовах динамічного ринку.

Монолітна архітектура історично залишається найпоширенішою моделлю, концептуальна сутність якої полягає у високому ступені зв'язності компонентів: бізнес-логіка, інтерфейс користувача та шар доступу до даних функціонують як єдиний, неподільний програмний модуль. Усі компоненти системи розробляються, компілюються та розгортаються як єдиний артефакт, що на

початкових етапах життєвого циклу проєкту має об'єктивні переваги. Централізація спрощує процеси налагодження, тестування та розгортання, а також забезпечує простоту управління транзакціями в єдиній базі даних. Проте зростання функціональності та обсягів даних неминуче призводить до значного ускладнення кодової бази, через що впровадження інноваційних рішень - таких як модулі доповненої реальності - у застарілу архітектуру часто стає технічно неможливим без ризику порушення стабільності критичних бізнес-процесів. Будь-яка локальна модернізація вимагає перезбірки та перезапуску всієї системи, а масштабування можливе лише вертикально через збільшення потужності сервера або шляхом повного клонування застосунку, що є ресурсозатратним та економічно неефективним підходом [17].

Мікросервісна архітектура представляє діаметрально протилежний підхід, передбачаючи декомпозицію системи на набір автономних служб, які групуються навколо певних бізнес-доменів. Кожен сервіс - чи то обробка замовлень, чи управління складськими запасами - функціонує як незалежний процес, має власну базу даних і взаємодіє з іншими компонентами через чітко визначені програмні інтерфейси (API) [31]. Така архітектура забезпечує високу гнучкість, дозволяючи використовувати різні технологічні стеки для вирішення специфічних завдань та незалежно масштабувати окремі модулі відповідно до навантаження. Водночас мікросервісний підхід характеризується високою операційною складністю, оскільки вимагає впровадження складних механізмів оркестрації контейнерів, налаштування виявлення сервісів та вирішення проблем узгодженості даних у розподіленому середовищі. Для проєктів середнього масштабу або MVP-рішень така складність виявляється надлишковою і призводить до невиправданого збільшення часових та фінансових витрат на розробку [32].

Обмеження вищезгаданих підходів зумовлюють перехід до концепції Headless Commerce, яка часто реалізується в рамках безсерверної (Serverless) моделі та передбачає повне архітектурне відокремлення клієнтської частини (Frontend) від серверної логіки (Backend). Взаємодія між рівнями відбувається

виключно через API, що, відповідно до цього принципу, дозволяє розробникам створювати гнучкі інтерфейси з використанням сучасних фреймворків, не впливаючи при цьому на стабільність серверних процесів (рис. 2.1).

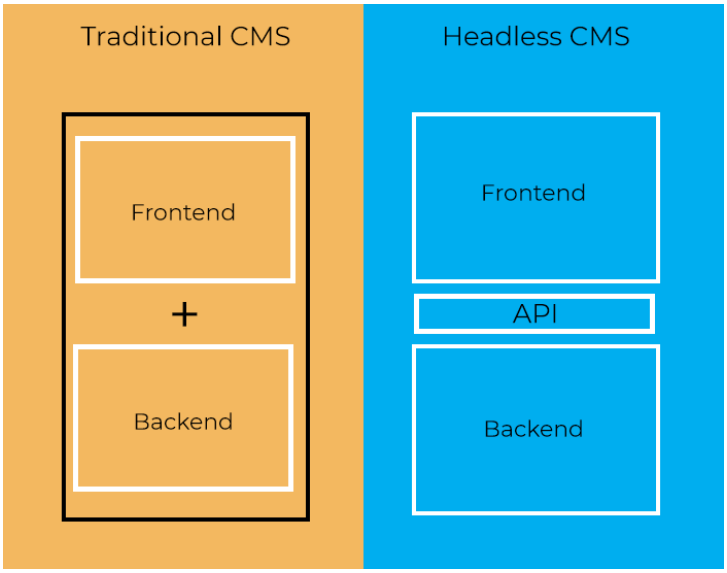


Рис. 2.1. Порівняння монолітної та headless архітектури [30].

Безсерверна модель передбачає делегування управління серверною інфраструктурою хмарному провайдеру, при цьому оплата здійснюється виключно за фактичний час виконання обчислювальних функцій. Результатом стає автоматичне масштабування системи та оптимізація фінансових витрат [19].

Вибір стратегії рендерингу контенту належить до критичних аспектів архітектурного проєктування - від цього рішення безпосередньо залежить користувацький досвід та позиції веб-ресурсу в пошукових системах. Необхідно розрізняти рендеринг на боці клієнта (Client-Side Rendering) та рендеринг на боці сервера (Server-Side Rendering) в контексті обраної архітектури. Перший підхід перекладає обчислювальне навантаження на пристрій користувача, що може призводити до уповільнення роботи на смартфонах з обмеженими ресурсами. Натомість другий підхід забезпечує формування повністю готової сторінки на сервері, гарантуючи швидке відображення контенту незалежно від потужності клієнтського пристрою. Для е-комерційних платформ, де швидкість

завантаження прямо корелює з показниками конверсії, найефективнішим виявляється гібридний підхід або статична генерація (Static Site Generation) - у цьому випадку сторінки формуються заздалегідь, що забезпечує миттєвий доступ користувачів до інформації.

У рамках даної кваліфікаційної роботи для реалізації веб-застосунку «spacia.ua» обрано гібридну сервіс-орієнтовану архітектуру на базі фреймворку Next.js. Подібний вибір дозволив гармонійно поєднати переваги різних підходів, одночасно нівелюючи притаманні їм недоліки. Технології статичної генерації та інкрементальної регенерації забезпечують високу продуктивність завантаження сторінок каталогу - цей параметр є критичним для утримання уваги користувача на платформі. Серверна бізнес-логіка реалізована через механізм API Routes, що дає змогу створювати модульну структуру бекенду без розгортання та адміністрування складної інфраструктури мікросервісів. Зовнішні інтеграції та логіка роботи з даними інкапсульовані у захищені серверні функції згідно з архітектурою рішення, тоді як клієнтська частина оптимізована для інтерактивної взаємодії, зокрема для роботи з модулями доповненої реальності. Досягнутий баланс між продуктивністю, безпекою даних, гнучкістю розробки та простотою подальшої підтримки програмного продукту повністю відповідає вимогам до сучасних конкурентоспроможних рішень у сфері електронної комерції.

2.3. Використання CI/CD та контейнеризації у процесі розробки

Написання вихідного коду в практиці розробки програмного забезпечення становить лише частину загального процесу створення продукту. Стабільність системи досягається через автоматизацію процесів тестування, збірки та розгортання - технології, відомі як CI/CD, - а також завдяки застосуванню контейнеризації. Впровадження цих стандартів суттєво прискорює випуск нових функцій, підвищуючи якість кінцевого рішення.

Концепція безперервної інтеграції (Continuous Integration, CI) спрямована на підтримку чистоти та цілісності програмного коду. Замість тривалої роботи в ізольованих гілках, що нерідко призводить до конфліктів при злитті, розробники додають зміни до основного репозиторію щоденно [34]. Ключовим інструментом тут виступає pull request - запит на зміни, який автоматично ініціює цикл перевірок: встановлення необхідних залежностей, статичний аналіз коду (зокрема, через ESLint) для виявлення стилістичних помилок, компіляція та запуск тестів різних рівнів, які відтворюють дії реального користувача.

Платформа GitHub Actions завоювала позиції одного з основних інструментів для реалізації CI завдяки тісній інтеграції з сервісом GitHub [35]. Робочий процес описується у файлах формату YAML, розміщених у директорії .github/workflows, що дозволяє налаштовувати послідовність кроків (jobs) на віртуальних машинах. Система забезпечує встановлення середовища виконання (Node.js), кешування модулів для прискорення збірки, перевірку коду лінтером. Можливість матричних збірок надає змогу протестувати додаток одночасно в різних операційних середовищах та версіях програмного забезпечення.

Безперервне розгортання (Continuous Deployment, CD) автоматизує передачу програмного коду безпосередньо у робоче середовище. Ефективним рішенням для проєктів на базі Next.js є сервіс Vercel (актуальна версія платформи станом на 2025-2026 рік), який пропонує механізм попереднього перегляду розгортань (Preview Deployments). Технологія автоматично генерує унікальну URL-адресу для кожної ітерації змін у коді (commit), дозволяючи проводити візуальне та функціональне тестування ізольованої версії продукту в реальному середовищі ще до злиття з основною гілкою проєкту [36]. Корисною функцією, доступною за потреби, є швидкий відкат (rollback), який дозволяє миттєво відновити попередню стабільну версію у разі виникнення технічних проблем.

Контейнеризація, зокрема технологія Docker, забезпечує стабільну роботу застосунку незалежно від середовища виконання [37]. Docker об'єднує код, необхідні бібліотеки та системні налаштування в єдиний ізольований контейнер.

Багатоетапна збірка (multi-stage build) дає змогу спочатку підготувати артефакти у повному середовищі розробки, після чого перенести лише необхідні файли у фінальний компактний образ - це оптимізує використання ресурсів. Інструмент Docker Compose дозволяє розробнику однією командою запустити всю необхідну інфраструктуру, включаючи базу даних та інші допоміжні сервіси.

Великі системи, що складаються з численних контейнерів, потребують інструментів оркестрації - наприклад, Kubernetes [38]. Ця система керує життєвим циклом контейнерів (подів), автоматично перезапускає їх у разі збоїв, регулює їх кількість відповідно до поточного навантаження. Втім, для проєктів масштабу spacia.ua впровадження Kubernetes може виявитися надмірно складним завданням, яке не відповідає реальним потребам системи.

У межах даної роботи обрано більш раціональний підхід, який краще відповідає задачам кваліфікаційної роботи: хмарна платформа самостійно керує інфраструктурою, забезпечуючи роботу HTTPS, CDN та моніторинг системи. Проєкт отримує високий рівень надійності без залучення додаткових ресурсів для налаштування складних серверних кластерів. Поєднання хмарної системи контролю версій GitHub та хмарного сервісу Vercel забезпечує швидкий шлях від написання коду до оновлення інтерфейсу користувача - кожна зміна проходить автоматичну верифікацію.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ Е-КОМЕРЦІЙНОГО ЗАСТОСУНКУ

3.1. Проєктування архітектури та структура системи

Проєктування архітектури належить до найвідповідальніших етапів розробки - саме воно визначає структуру системи, принципи взаємодії її частин та технічні шляхи реалізації бізнес-завдань. Архітектурне рішення для платформи spacia.ua базується на поєднанні сучасних вебтехнологій, при цьому основний акцент зроблено на забезпеченні високої швидкодії, можливості подальшого розширення системи та зручності процесу розробки.

В основу проєкту покладено клієнт-серверну модель, реалізовану за допомогою фреймворку Next.js на базі бібліотеки React. Обраний інструментарій дозволив об'єднати фронтенд та бекенд в єдину екосистему - це значно спростило процеси написання коду та розгортання застосунку. Клієнтська частина функціонує безпосередньо у браузері користувача, де відбувається рендеринг інтерфейсу та обробка дій. Застосунок побудовано як Single Page Application (SPA), однак з активним використанням технологій серверного рендерингу (SSR) та статичної генерації сторінок (SSG) - такий підхід позитивно впливає на SEO та швидкість відображення контенту.

Серверна частина реалізована через механізм Next.js API Routes, що дозволило створити серверні точки доступу (ендпоінти) безпосередньо всередині проєкту без розгортання окремого бекенд-сервера. API-маршрути виконують роль прошарку бізнес-логіки: вони обробляють запити від клієнта, взаємодіють із зовнішніми сервісами, повертають дані у форматі JSON. Важливою особливістю є те, що при розміщенні на хмарній платформі Vercel кожен такий маршрут автоматично трансформується в ізольовану безсерверну (serverless) функцію. Товарні дані для поточної версії (MVP) зберігаються локально у файлі lib/data.js.

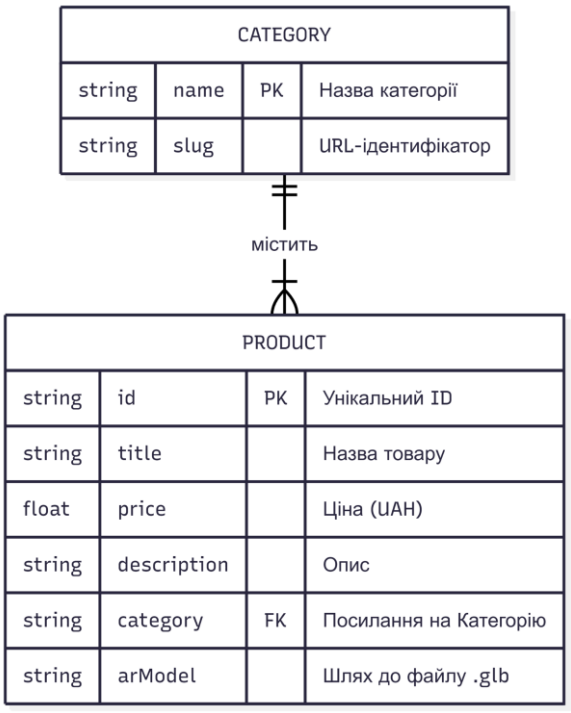


Рис. 3.1. Логічна модель даних системи
**розроблено автором*

Незважаючи на використання файлової системи для зберігання інформації на етапі MVP, структура даних була спроєктована за принципами реляційної моделі (рис. 3.1). Це забезпечує цілісність посилань між сутностями PRODUCT та CATEGORY, а також дозволяє масштабувати систему шляхом міграції на SQL-орієнтовану базу даних (наприклад, PostgreSQL) без зміни бізнес-логіки застосунку. Атрибут arModel виконує роль покажчика на оптимізовані 3D-ресурси, відокремлюючи метадані товару від важкого медіаконтенту.

Фронтенд застосунку побудовано на чіткому компонентному підході, де інтерфейс розділений на незалежні модулі, які можна використовувати повторно. Ієрархія основних компонентів організована за їхнім функціональним призначенням:

- Кореневий макет (app/layout.tsx): визначає загальну структуру сторінок, метадані застосунку та підключення контекстних провайдерів.

- Навігаційна панель (Header): реалізована як зафіксований (sticky) елемент, що забезпечує постійний доступ до меню категорій та кошика.
- Картка товару (ProductCard): універсальний адаптивний компонент для візуалізації інформації про товар у каталозі.
- AR-модуль (ARView): забезпечує інтеграцію вебкомпонента <model-viewer> для перегляду 3D-моделей та взаємодії з камерою пристрою.

Архітектура клієнтської частини (рис. 3.2) базується на патерні Context Provider. Кореневий компонент RootLayout ініціалізує глобальний стан кошика (CartProvider), що робить його доступним для всіх дочірніх компонентів без необхідності "наскрізної передачі параметрів" (props drilling).

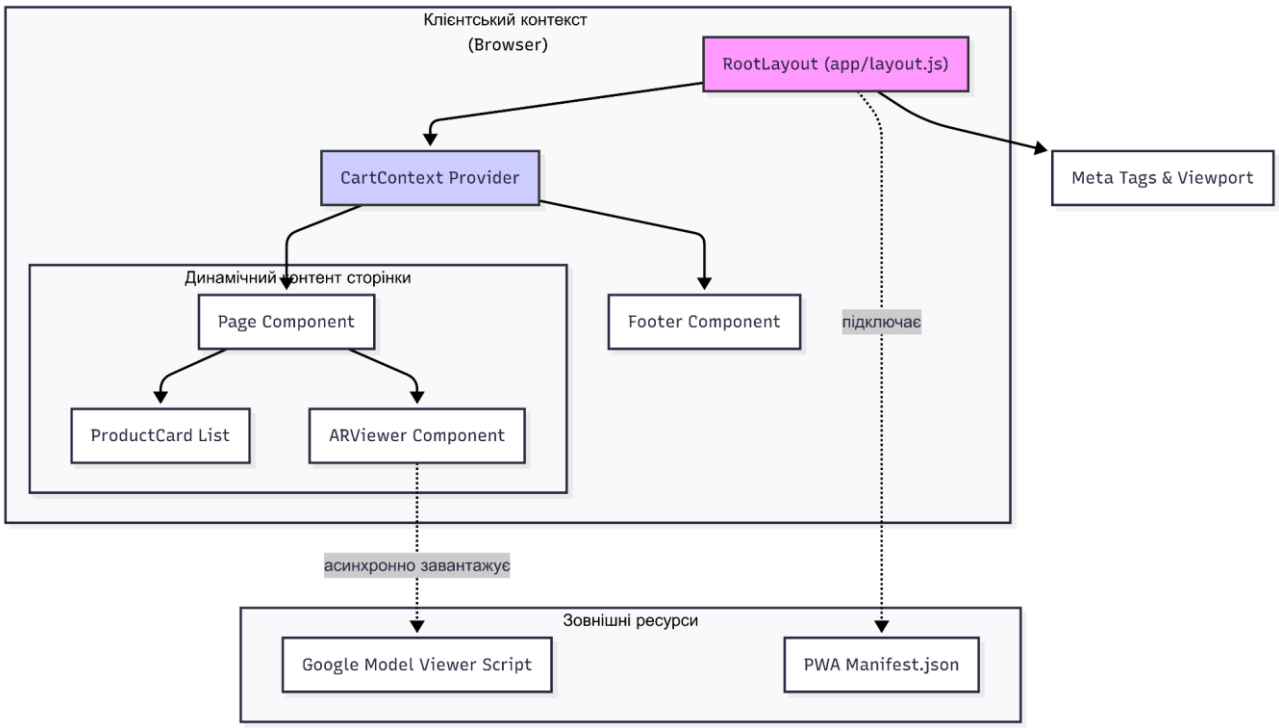


Рис. 3.2. Компонентна архітектура застосунку
**розроблено автором*

Асинхронне підключення зовнішніх скриптів (Google Model Viewer) реалізовано на рівні конкретних компонентів (ARViewer), що оптимізує час завантаження перших компонентів (FCP).

Для управління глобальним станом застосунку, зокрема вмістом кошика, було застосовано React Context API. Це рішення дозволило уникнути передачі

даних через довгі ланцюжки компонентів (props drilling), надаючи доступ до функцій маніпуляції товарами з будь-якого вузла системи. Маршрутизація проєкту побудована на базі сучасної системи Next.js App Router, де структура сторінок визначається файловою системою, а динамічні параметри (наприклад, ID товару в app/item/[productId]) обробляються автоматично.

Організація файлової системи проєкту відповідає загальноприйнятим стандартам фреймворку Next.js:

- app/: містить основну логіку сторінок, макети та налаштування маршрутів.
- components/: бібліотека всіх UI-компонентів системи.
- lib/: включає допоміжні утиліти, опис типів даних (TypeScript interfaces) та файли зі статичними даними.
- public/: репозиторій для медіафайлів (зображень та 3D-моделей), які обслуговуються через мережу доставки контенту (CDN).
- Конфігураційні файли: включають налаштування середовища збірки (next.config.js), стилізації (tailwind.config.js) та параметри типізації (tsconfig.json) (рис. 3.3).



Рис. 3.3. Структура файлів проєкту в середовищі розробки
** авторська розробка*

У межах системи було імплементовано гібридну модель рендерингу, що адаптується під тип контенту:

- SSG (Static Site Generation): Застосовано для сторінок категорій. HTML генерується на етапі запуску, що забезпечує миттєву віддачу контенту.
- SSR (Server-Side Rendering): Використовується для сторінок товарів, де критична актуальність даних (ціна, наявність).
- CSR (Client-Side Rendering): Задіяно для високоінтерактивних елементів (кошик, AR-перегляд), що залежать від дій користувача.

Оптимізація медіаконтенту посідає важливе місце в архітектурі проєкту [41]. Стандартний компонент Next.js Image автоматизує генерацію зображень у форматах WebP та AVIF, забезпечуючи повну адаптивність графіки під різноманітні типи пристроїв. Впроваджено механізм «ледачого завантаження» (Lazy loading), а блюр-плейсхолдери резервують простір під зображення до моменту його повного завантаження. Такий підхід запобігає зміщенню макету, підтримуючи стабільність показника CLS.

Серверні маршрути (API Routes) забезпечують взаємодію із зовнішніми сервісами, при цьому ключі доступу зберігаються виключно на стороні сервера. Платіжний шлюз Stripe імплементовано через модель створення сесій на сервері - ймовірність компрометації конфіденційних даних під час транзакцій мінімізується [42]. Технологія Google Model Viewer лягла в основу візуалізації об'єктів у доповненій реальності (AR): інструмент автоматично розпізнає тип пристрою користувача, обираючи програмний рушій ARCore для Android або ARKit для iOS [43, 44].

Багаторівнева структура характеризує систему захисту інформації в застосунку spacia.ua. Примусове використання протоколу HTTPS із сертифікатами Let's Encrypt гарантує безпеку транспортного рівня. На рівні застосунку налаштовані політики безпеки контенту (CSP) - вони захищають від XSS-атак, водночас ретельна валідація вхідних даних на сервері усуває ризики програмних ін'єкцій. Чутливі конфігураційні параметри винесено у змінні оточення (Environment Variables) з чітким розмежуванням за зонами доступу, що унеможливлює їх потрапляння у відкритий клієнтський код.

Результати проєктування демонструють збалансованість обраної архітектурної моделі. Поєднання гнучкості мікросервісного підходу (через serverless-функції) та зручності розробки монолітних систем гарантує високий рівень безпеки, повну відповідність сучасним стандартам продуктивності Web Vitals.

3.2. Зміст процесу розробки функціоналу е-комерційного застосунку

Функціонал е-комерційного застосунку spacia.ua імплементовано згідно з принципами інкрементальної розробки - система нарощувала можливості поетапно, від базової архітектури до складних інтерактивних модулів. Технічний старт проєкту реалізовано через ініціалізацію середовища Next.js із застосуванням TypeScript, інтегровано CSS-фреймворк Tailwind для стандартизації стилізації компонентів та забезпечення адаптивності інтерфейсу [39].

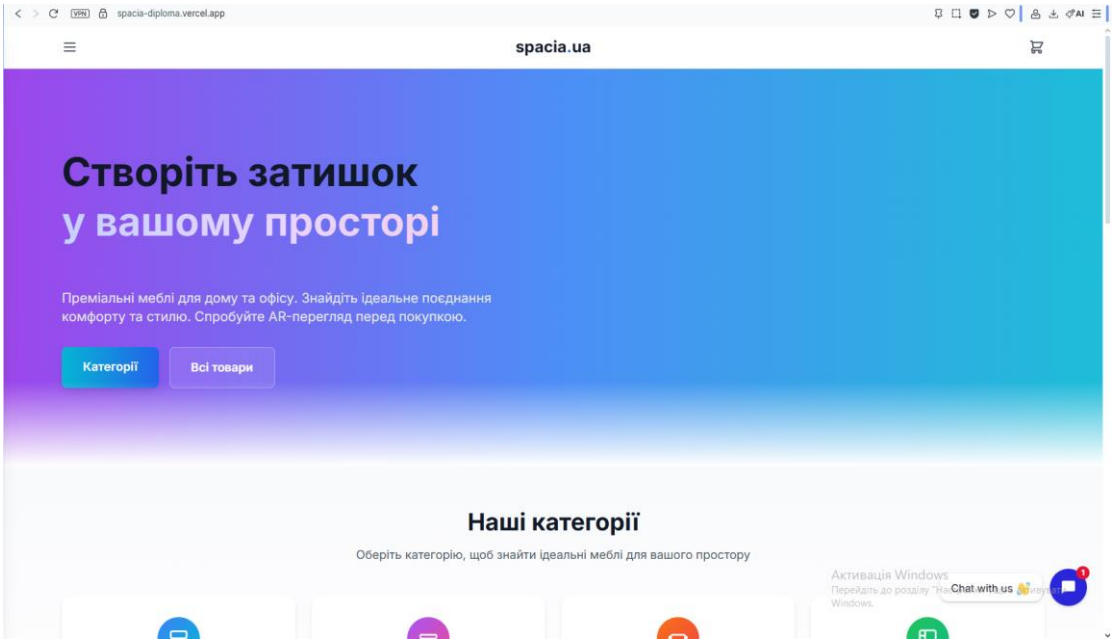


Рис. 3.4. Інтерфейс головної сторінки
* *авторська розробка*

Структура вихідного коду проєкту базується на принципі чіткого розподілу обов'язків між окремими директоріями. Зокрема, папка app містить логіку маршрутизації та основні сторінки застосунку, тоді як у директоріях components та lib зосереджені компоненти інтерфейсу для повторного використання та допоміжні функції. Повний лістинг програмного коду розроблених модулів наведено у Додатку В.

Одночасно з налаштуванням архітектури було впроваджено систему контролю версій Git. Для запобігання потраплянню системних файлів та

конфіденційних ключів до репозиторію було належним чином налаштовано файл .gitignore. Процес безпосередньої розробки інтерфейсу розпочався зі створення наскрізних елементів навігації. Головним серед них став компонент Header, який реалізовано як закріплену панель (sticky element) для забезпечення постійного доступу до основних розділів сайту та кошика. Також було реалізовано адаптивність цього модуля: при використанні на мобільних пристроях навігаційне меню трансформується у компактне «бургер-меню».

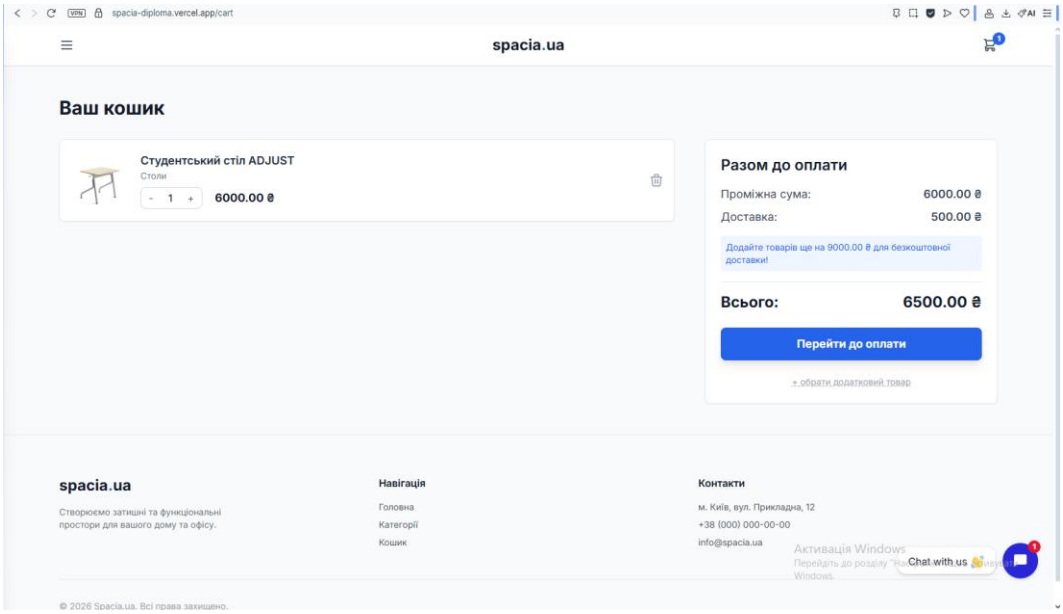


Рис. 3.5. Сторінка з кошиком

** авторська розробка*

Компонент ProductCard лежить в основі вітрини - саме він забезпечує візуалізацію окремих товарів. Автоматична оптимізація графічних файлів досягається через вбудований інструмент Next.js Image. Інтерфейс стає живішим завдяки ефектам при наведенні курсора (hover-ефектам).

Вся логіка покупок сконцентрована в CartContext, побудованому на React Context API. Такий архітектурний підхід відкриває можливість змінювати склад кошика з будь-якої частини сайту: додавати позиції, видаляти їх, перераховувати суму. Сторінка оновлюється миттєво, без повного перезавантаження. Інформація

про вибрані товари зберігається в локальному сховищі браузера - навіть після закриття вкладки дані не зникають.

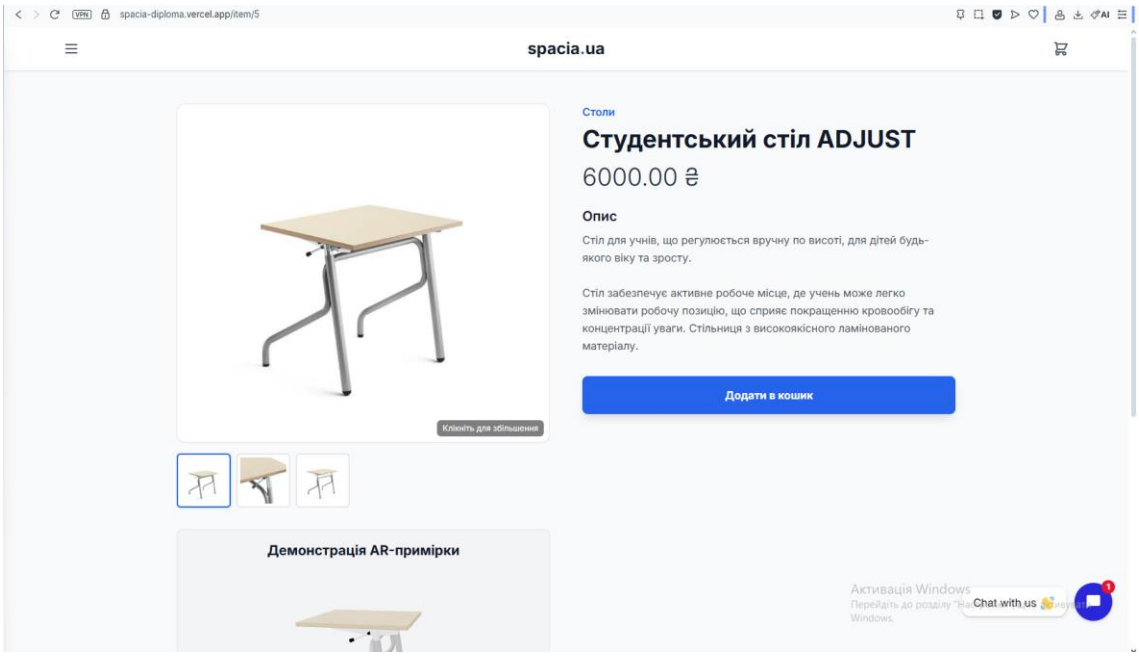


Рис. 3.6. Сторінка товару

** авторська розробка*

Система динамічної маршрутизації забезпечує наповнення сторінок функціоналом. Головна сторінка виконує роль основної вітрини застосунку - тут поєднано презентаційну Hero-секцію, навігаційний блок категорій та перелік рекомендованих товарів. Сторінки категорій генеруються статично (SSG) на основі параметрів URL, завдяки чому контент завантажується практично миттєво. Детальна інформація про конкретний виріб (за шляхом app/item/[productId]) підвантажується динамічно. Візуальна частина отримала особливу увагу: галереї товарів мають плавні переходи, блок із технічними характеристиками структуровано за допомогою семантичних тегів - це покращує загальну доступність інтерфейсу.

Важливою частиною проєкту стала розробка модуля ARView для роботи з технологіями доповненої реальності (AR). Використання вебкомпонента <model-viewer> дозволило впровадити підтримку стандартів WebXR, завдяки чому користувачі можуть переглядати 3D-моделі товарів у форматі GLB

безпосередньо у власному інтер'єрі. Додаток автоматично визначає, чи підтримує пристрій клієнта технології ARCore або ARKit, і відповідно до цього адаптує роботу інтерфейсу. Щоб забезпечити високу швидкість роботи та стабільність, 3D-моделі пройшли етап попередньої оптимізації, яка включала компресію текстур та спрощення геометрії об'єктів.

Алгоритм на рис. 3.7 розв'язує проблему дублювання ресурсів під час навігації користувача.

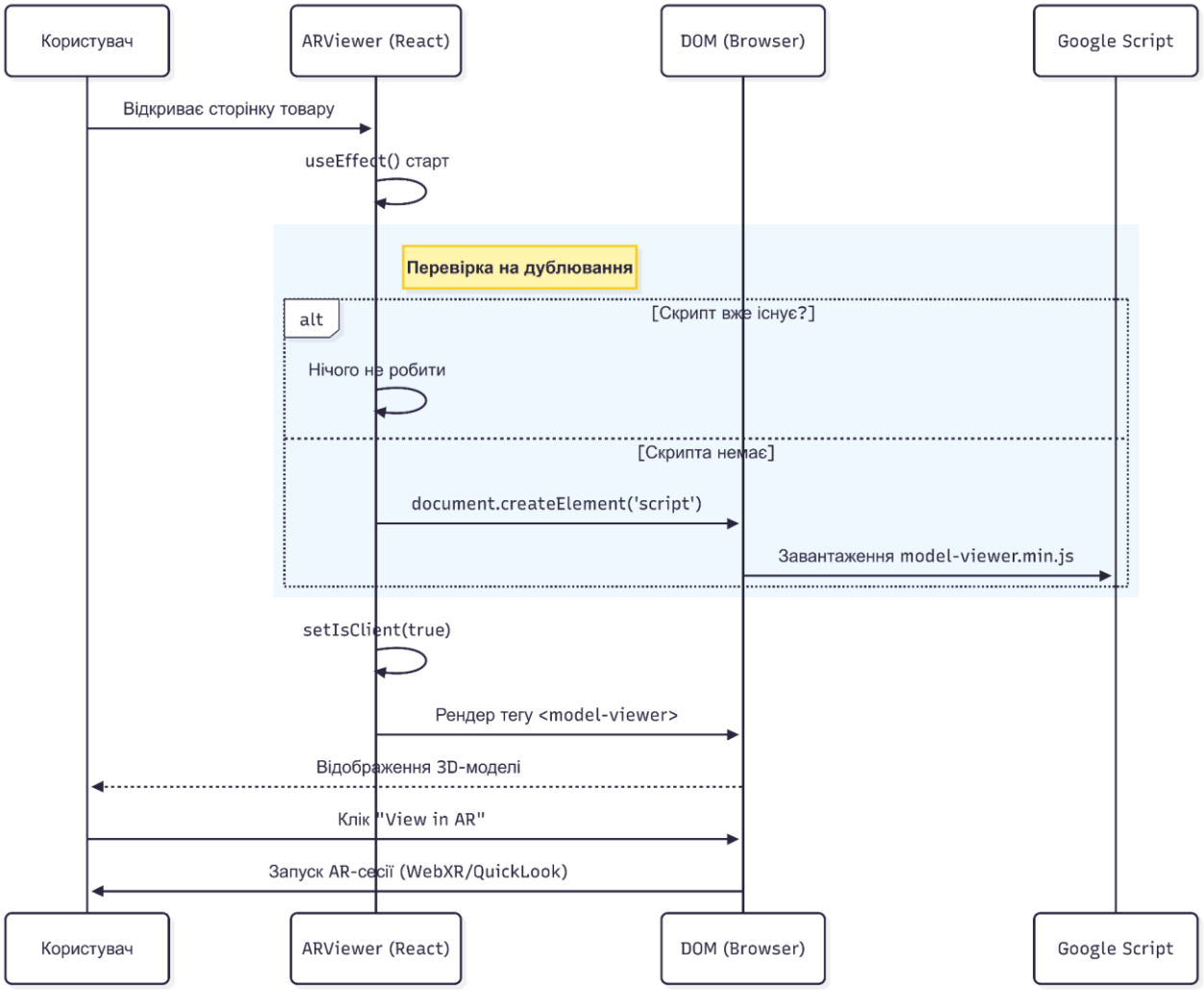


Рис. 3.7. Алгоритм асинхронної ініціалізації AR-сесії.
** авторська розробка*

Застосування хука useEffect обмежує ініціалізацію WebXR клієнтським середовищем браузера - серверний рендеринг (SSR) у Next.js не має доступу до об'єкта document, тому без цього обмеження виникають помилки гідратації (Hydration Error). Зовнішня бібліотека завантажується виключно за необхідності, що знижує навантаження на мережу.

Технологія WebXR у поєднанні з компонентом <model-viewer> надає користувачу можливість візуалізувати тривимірну модель товару в реальному масштабі безпосередньо у власному приміщенні. Скріншот демонструє коректне розпізнавання системою площини підлоги (Surface Tracking) - модель студентського столу накладається на поверхню, генеруються реалістичні тіні для створення ефекту присутності (див. Додаток А). Покупець отримує змогу оцінити габарити меблів та їх відповідність інтер'єру ще до здійснення покупки - це одна з ключових переваг розробленого застосунку перед класичними інтернет-магазинами.

Для проведення розрахунків у системі було реалізовано інтеграцію з платіжною платформою Stripe через використання протоколу Payment Intents API, на основі використання [42].

Алгоритм взаємодії передбачає, що клієнтський застосунок ініціює запит на сервер для створення наміру платежу (PaymentIntent), отримуючи у відповідь унікальний секретний ключ (client_secret). Цей ключ використовується бібліотекою Stripe.js на стороні клієнта для безпечної передачі платіжних реквізитів напряму до шлюзу провайдера, оминаючи сервер застосунку, що забезпечує відповідність вимогам стандарту PCI DSS. Після успішної обробки транзакції інтерфейс користувача виводить модальне діалогове вікно (інтерактивний елемент, що блокує взаємодію з основним контентом до виконання певної дії) з підтвердженням статусу оплати та унікальним ідентифікатором замовлення, а дані про покупку асинхронно фіксуються в базі даних системи.

Для того, щоб зробити взаємодію з вебресурсом максимально схожою на використання нативних мобільних програм, було впроваджено технологію Progressive Web App (PWA) (див. Додаток Б). Основні налаштування інтерфейсу та іконок містяться у файлі маніфесту (manifest.json), що дозволяє користувачам встановлювати сайт безпосередньо на домашній екран свого пристрою [45].

Завдяки роботі Service Worker забезпечується кешування важливих файлів. Для динамічного контенту обрано стратегію «спочатку мережа» (network-first), а для статичних ресурсів — активне кешування. Такий підхід дозволяє платформі залишатися частково доступною навіть у разі тимчасової відсутності інтернет-з'єднання.

Фінальним етапом розробки стала всебічна оптимізація швидкодії та SEO-параметрів. Завдяки впровадженню «лінивого» завантаження зображень (Lazy loading), автоматичному розподілу коду (code splitting) та попередньому завантаженню посилань (prefetching), вдалося досягти високих значень за метриками Core Web Vitals [40], зокрема Largest Contentful Paint (LCP) (час відображення основного елемента сторінки), Interaction to Next Paint (INP) (показник візуальної стабільності) та Cumulative Layout Shift (CLS) (реакція інтерфейсу на дії користувача). Окрім семантичної верстки, сторінки було доповнено структурованими даними у форматі JSON-LD. Це допомагає пошуковим системам краще індексувати товари та дозволяє відображати розширені сніпети у результатах видачі.

3.3. Тестування, розгортання та оцінка ефективності проєкту

Забезпечення високої якості програмного продукту — це не лише перевірка функцій, а й контроль його надійності та зручності для користувача. Процес верифікації платформи spacia.ua базувався на стратегії «піраміди тестування», яка поєднує автоматичні перевірки з ретельним ручним аудитом.

Основою перевірки стало модульне тестування (unit testing), яке дозволило ізольовано протестувати окремі функції системи. Зокрема, було перевірено логіку фінансових розрахунків та коректність валідації введених даних. Для React-компонентів використовувалися спеціальні заглушки (моки), що дозволило імітувати зовнішні дані та перевіряти відображення інтерфейсу за різних умов. Інтеграційні тести зосередилися на взаємодії модулів - наприклад, на передачі даних через Context API під час роботи з кошиком. Серверна частина (API Routes) тестувалася шляхом імітації відповідей від платіжного шлюзу Stripe, що дозволило відпрацювати сценарії з помилками без проведення реальних платежів.

Паралельно проводилось тестування роботи застосунку в різних браузерах. Тестування на мобільних пристроях Apple (iOS/Safari) виявило специфіку роботи AR-модуля: на відміну від Android-пристроїв, які підтримують стандарт WebXR, браузер Safari вимагає використання нативного механізму Quick Look.

Для забезпечення кросплатформності у компонент <model-viewer> було додано атрибут конфігурації ar-modes, який автоматично визначає тип пристрою та перемикає рушій візуалізації (WebXR або Quick Look), гарантуючи стабільний запуск AR-сесії на обох платформах. Після цього, адаптивність дизайну перевірялась на реальних смартфонах під управлінням iOS та Android, в результаті чого підтвердилась стабільність інтерфейсу при зміні розмірів екрана.

Окремо досліджувалася робота сайту при повільному інтернет-з’єднанні (імітація 3G). Це підтвердило ефективність використання «скелетних» екранів (skeletons) та поступового завантаження зображень для утримання уваги відвідувача.

Інфраструктура для розгортання проєкту була побудована на базі екосистеми Vercel з автоматизацією через GitHub (CI/CD). Використання Edge Network дозволяє кешувати статичні файли на CDN-вузлах, що значно зменшує затримки для користувача. Механізм попереднього перегляду (preview deployments) дозволив тестувати кожну зміну в окремому середовищі перед її

публікацією, а функція миттєвого відкату (rollback) гарантує швидке відновлення стабільної версії у разі збою (рис. 3.8).

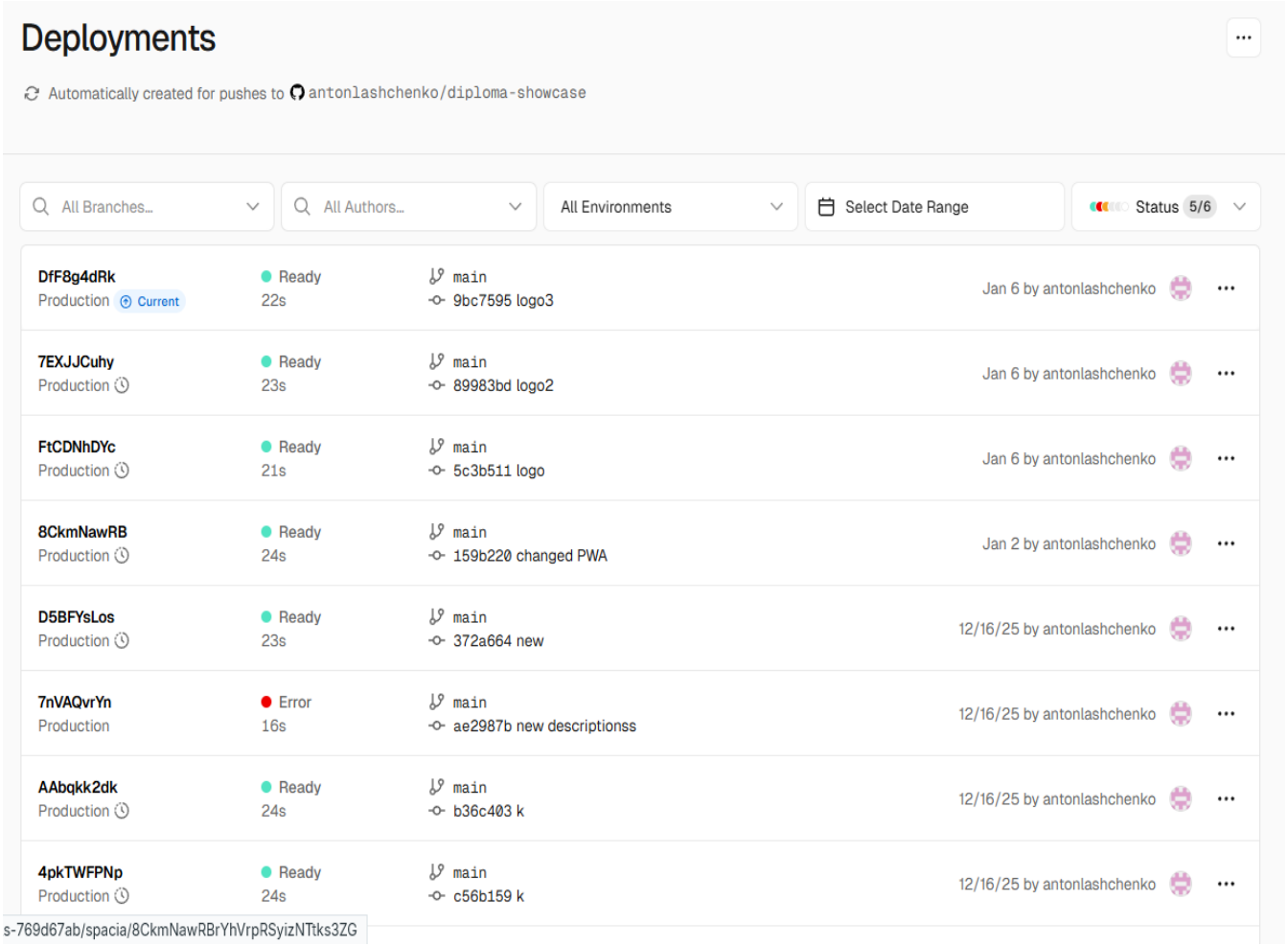


Рис. 3.8. Дешборд в екосистемі Vercel

* *авторська розробка*

Для контролю швидкості роботи сайту використовувався відомий сервіс Google PageSpeed Insights. На сьогодні цей сервіс вважається одним з найбільш інформативних для аналізу технічного стану вебресурсів [46].

Отримані практичні дані підтвердили високу якість оптимізації за метриками Core Web Vitals (рис. 3.9):

- Largest Contentful Paint (LCP) — час рендерингу найбільшого елемента становить 0,6 секунди, що стало можливим завдяки статичній генерації сторінок (SSG).

- Cumulative Layout Shift (CLS) — показник зміщення макету зведений до нуля, оскільки для всього медіаконтенту заздалегідь зарезервовано місце у верстці.

Загальна оцінка продуктивності (Performance Score) сягнула 100 балів зі 100. Це значно вищий показник порівняно з традиційними системами управління контентом (CMS), які зазвичай є більш громіздкими.

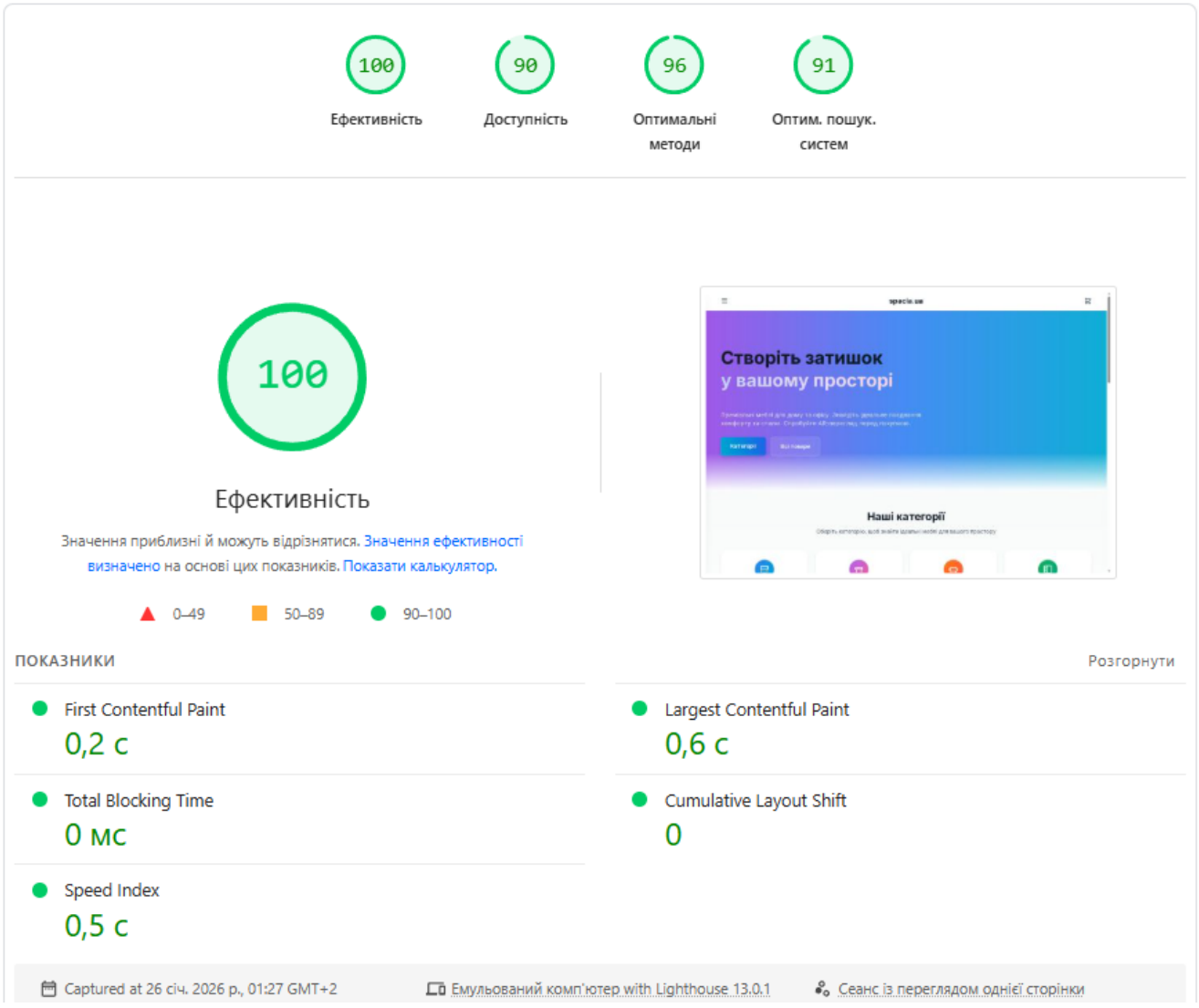


Рис. 3.9. Результати ефективності
** авторська розробка*

Порівняльний аналіз демонструє суттєві переваги обраної архітектури у швидкодії та можливостях подальшого масштабування. Стандарти PWA роблять

взаємодію із сайтом такою ж зручною, як і з нативним мобільним застосунком, тоді як AR-модуль створює унікальні переваги для користувачів.

Безпека платформи гарантується комплексом рішень: примусове використання протоколу HTTPS, налаштування політик CSP, передача обробки платіжних даних сертифікованому провайдеру Stripe. Створена архітектура становить гнучку базу - у майбутньому вона дозволить без зайвих труднощів інтегрувати повноцінну базу даних або системи машинного навчання для персоналізації пропозицій.

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено комплексне дослідження, спрямоване на аналіз сучасних тенденцій у сфері розробки програмного забезпечення для електронної комерції, а також здійснено практичну реалізацію повнофункціонального застосунку spacia.ua, який слугує демонстрацією реалізації цих тенденцій у реальному проекті. Теоретична база дослідження дозволила визначити основні принципи функціонування систем електронної комерції та шість основних тенденцій розвитку галузі: мобільна адаптація, технології PWA та застосування доповненої реальності, хмарні обчислення, шаблони мікросервісів та вдосконалені протоколи кібербезпеки.

Порівняльний аналіз інструментів розробки виявив високу різноманітність існуючих технологічних рішень. Хоча готові платформи класу Shopify або Magento пропонують швидкий старт, вони накладають значні обмеження на налаштування. На відміну від кастомної розробки зі стеком React та Next.js, забезпечує гнучкість та продуктивність, необхідні для задоволення унікальних бізнес-вимог. Архітектурний пошук не виявив універсальної закономірності: моноліт виправданий у невеликих проектах, мікросервіси - у корпоративних системах, а комбінований, безсерверний метод виявився найбільш ефективним у середньому сегменті.

Практична реалізація проєкту spacia.ua успішно продемонструвала поєднання досліджених технологій. Next.js використовується в архітектурі системи для забезпечення інтегрованого зв'язку між фронтендом та бекендом. Tailwind CSS забезпечує мобільну адаптивність для коректного відображення інтерфейсу на всіх пристроях, тоді як концепція Progressive Web App переносить користувацький досвід на рівень нативних додатків з можливістю встановлення та офлайн-доступу. Інтеграція AR-модуля на базі Google Model Viewer належить до найважливіших функцій - вона дозволяє переглядати 3D-моделі меблів у власному інтер'єрі, що значно покращує процес прийняття рішень про покупку.

Проєкт розгорнуто в хмарі на платформі Vercel, де контент автоматично постачається через власну CDN. Безсерверні функції Next.js у бекенді реагують

на маршрути API, ефективно розподіляючи навантаження. Безпека забезпечується кількома заходами: HTTPS-шифрування, політика безпеки контенту, інтеграція платіжного шлюзу Stripe - це запобігає обміну конфіденційною інформацією через клієнтський код.

Розроблене рішення пройшло багаторівневе тестування, яке підтвердило кросбраузерну сумісність та стабільність на мобільних платформах Android і iOS. Емпіричні дані Google PageSpeed Insights демонструють високу ефективність застосованих оптимізацій: застосунок набрав 100 балів для версії настільних комп'ютерів та 98 балів для мобільної версії. Показники Core Web Vitals - LCP за 0,6 секунди та нульовий CLS - підтверджують перевагу застосованої архітектури над традиційними шаблонними рішеннями.

За результатами дослідження було систематизовано основні тенденції, обґрунтовано вибір технологічного стеку та створено продукт, що відповідає сучасним стандартам якості коду та продуктивності. Практична цінність роботи полягає у створенні відкритої реалізації застосунку електронної комерції, який можна масштабувати. Застосування алгоритмів машинного навчання для персоналізації контенту, інтеграція розширеної бази даних та впровадження системи автентифікації - деякі з майбутніх перспектив, пов'язаних з проєктом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laudon K. C., Traver C. G. E-commerce 2021-2022: business, technology, society. 16th ed. Harlow : Pearson, 2021. URL: https://api.pageplace.de/preview/DT0400.9781292343211_A39745346/preview-9781292343211_A39745346.pdf (дата звернення: 20.12.2025).

2. Лащенко А. О. Огляд сучасних тенденцій розробки ПЗ для е-комерції /А. О. Лащенко, І. Д. Погорєловська //Економіка і бізнес: нові реалії, технологічні тренди, цифрові інновації. Economy and business: new realities, technological trends, digital innovations: збірник матеріалів Міжнародної науково-практичної конференції здобувачів вищої освіти та молодих вчених (Львів-Ірпінь, 15 травня 2025 р.). [Елекронне вид.]. Львів: СПДФО Омельченко В. Г. 2025, с.188-191. – Режим доступу: https://financial.lnu.edu.ua/wp-content/uploads/2025/06/Zbirnyk-konferentsii-15-travnia-2025.pdf (дата звернення: 20.12.2025)

3. Laudon K. C., Traver C. G. E-commerce: Business, Technology, Society. 2019. URL: https://studylib.net/doc/27021184/kenneth-c.-laudon—carol-guercio-traver---e-commerce---bu (дата звернення: 20.12.2025).

4. Digital Transformation in Business: A Comparative Study of E-Commerce Platforms. International Journal of Advanced Research. 2025. URL: https://www.researchgate.net/publication/394662138_DIGITAL_TRANSFORMATION_IN_BUSINESS_A_COMPARATIVE_STUDY_OF_E-COMMERCE_PLATFORMS (дата звернення: 20.12.2025).

5. Digital Economy Report 2024: Shaping an environmentally sustainable and inclusive digital future (Overview). UNCTAD. 2024. URL: https://unctad.org/system/files/official-document/der2024_overview_en.pdf (дата звернення: 20.12.2025).

6. Mobile e-commerce sales worldwide from 2014 to 2027 [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/ (дата звернення: 20.12.2025).

7. Google Developers. What are Progressive Web Apps? [Електронний ресурс]. – Режим доступу: https://web.dev/explore/progressive-web-apps (дата звернення: 20.12.2025).

8. IBM. AI in retail: Transforming the customer experience [Електронний ресурс]. – Режим доступу: https://www.ibm.com/industries/retail/artificial-intelligence (дата звернення: 20.12.2025).

9. Shopify. The Future of Commerce: Trends Report [Електронний ресурс]. – Режим доступу: https://www.shopify.com/enterprise/the-future-of-commerce (дата звернення: 20.12.2025).

10. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 20.12.2025).

11. Dragoni N., Giallorenzo S., Lafuente A. L. Microservices: yesterday, today, and tomorrow [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/315664446_Microservices_yesterday_today_and_tomorrow (дата звернення: 20.12.2025).

12. IBM. IaaS, PaaS and SaaS [Електронний ресурс]. – Режим доступу: https://www.ibm.com/think/topics/iaas-paas-saas (дата звернення: 20.12.2025).

13. Shopify. Augmented Reality in Ecommerce: How AR is Changing Shopping [Електронний ресурс]. – Режим доступу: https://www.shopify.com/enterprise/blog/augmented-reality-in-commerce (дата звернення: 20.12.2025).

14. eCommerce Fastlane. Чому варто звернути увагу на Headless Commerce? [Електронний ресурс]. – Режим доступу:

https://ecommercefastlane.com/uk/why-should-you-care-about-headless-commerce (дата звернення: 20.12.2025).

15. Nguyen T. S. The Rise Of The Pragmatic Consumer: Reshaping Retail In 2025. Forbes. 2025. [Електронний ресурс]. – Режим доступу: https://www.forbes.com/councils/forbesbusinesscouncil/2025/01/03/the-rise-of-the-pragmatic-consumer-reshaping-retail-in-2025/ (дата звернення: 20.12.2025).

16. Бондаренко В. М., Омельяненко О. В. Цифровий маркетинг сьогодення: переваги та недоліки. Економіка та суспільство. 2024. Вип. 67. URL: https://economyandsociety.in.ua/index.php/journal/article/view/4681/4623 (дата звернення: 20.12.2025).

17. Richards M. Software Architecture Patterns. O'Reilly Media, 2015. 48 p. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ (дата звернення: 20.12.2025).

18. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 616 p. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/ (дата звернення: 20.12.2025).

19. Sbarski P., Kroonenburg S. Serverless Architectures on AWS. Manning Publications, 2017. 354 p. [Електронний ресурс]. – Режим доступу: https://www.manning.com/books/serverless-architectures-on-aws (дата звернення: 20.12.2025).

20. Shopify. About Shopify [Електронний ресурс]. – Режим доступу: https://www.shopify.com/about (дата звернення: 20.12.2025).

21. Shopify. Shopify Plus: Enterprise Ecommerce Platform [Електронний ресурс]. – Режим доступу: https://www.shopify.com/plus (дата звернення: 20.12.2025).

22. Adobe. Adobe Commerce (Magento) [Електронний ресурс]. – Режим доступу: https://business.adobe.com/products/magento/magento-commerce.html (дата звернення: 20.12.2025).

23. WooCommerce. The most customizable eCommerce platform [Електронний ресурс]. – Режим доступу: https://woocommerce.com/ (дата звернення: 20.12.2025).

24. React. The Library for Web and Native User Interfaces [Електронний ресурс]. – Режим доступу: https://react.dev/ (дата звернення: 20.12.2025).

25. Wieruch R. The Road to React [Електронний ресурс]. – Режим доступу: https://www.roadtoreact.com/ (дата звернення: 20.12.2025).

26. Next.js Documentation [Електронний ресурс]. – Режим доступу: https://nextjs.org/docs (дата звернення: 20.12.2025).

27. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs/ (дата звернення: 20.12.2025).

28. Eagles L. Build an API with Serverless Functions in Next.js [Електронний ресурс]. – Режим доступу: https://blog.logrocket.com/build-api-serverless-functions-next-js/ (дата звернення: 20.12.2025).

29. How to Choose Between SaaS, Open-Source, and Custom E-Commerce Solutions [Electronic resource]. Namaa IT. 2025. URL: https://www.namaait.com/en/articles/97/how-to-choose-between-ecommerce-solutions (дата звернення: 20.12.2025).

30. Clean Commit. Traditional vs Headless CMS: Are Monolithic CMSs Dead? [Електронний ресурс]. – Режим доступу: https://cleancommit.io/blog/traditional-vs-headless-cms-are-monolithic-cmss-dead/ (дата звернення: 20.12.2025).

31. Fowler M. Microservices: A Definition of This New Architectural Term [Електронний ресурс]. – Режим доступу:

https://martinfowler.com/articles/microservices.html (дата звернення: 20.12.2025).

32. Microservices.io. Microservices Patterns [Електронний ресурс]. – Режим доступу: https://microservices.io/patterns/index.html (дата звернення: 20.12.2025).

33. AWS. Serverless Computing [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/serverless/ (дата звернення: 20.12.2025).

34. Humble J., Farley D. Continuous Delivery [Електронний ресурс]. – Режим доступу: https://continuousdelivery.com/ (дата звернення: 20.12.2025).

35. GitHub Actions Documentation [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/actions (дата звернення: 20.12.2025).

36. Vercel. Deploying Projects [Електронний ресурс]. – Режим доступу: https://vercel.com/docs/deployments (дата звернення: 20.12.2025).

37. Turnbull J. The Docker Book [Електронний ресурс]. – Режим доступу: https://dockerbook.com/ (дата звернення: 20.12.2025).

38. Burns B., Beda J. Kubernetes: Up and Running. 3rd ed. O'Reilly Media, 2022. 350 p. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/kubernetes-up-and/9781492046523/ (дата звернення: 20.12.2025).

39. Tailwind CSS Documentation [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs (дата звернення: 20.12.2025).

40. Google. Web Vitals [Електронний ресурс]. – Режим доступу: https://web.dev/articles/vitals (дата звернення: 20.12.2025).

41. DebugBear. Next.js Performance Optimization Guide [Електронний ресурс]. – Режим доступу: https://www.debugbear.com/blog/nextjs-performance (дата звернення: 20.12.2025).

42. Stripe. API Reference [Електронний ресурс]. – Режим доступу: https://docs.stripe.com/api (дата звернення: 20.12.2025).

43. Google Developers. ARCore [Електронний ресурс]. – Режим доступу: https://developers.google.com/ar (дата звернення: 20.12.2025).

44. Apple Developer. ARKit [Електронний ресурс]. – Режим доступу: https://developer.apple.com/augmented-reality/arkit/ (дата звернення: 20.12.2025).

45. W3C. Web App Manifest [Електронний ресурс]. – Режим доступу: https://www.w3.org/TR/appmanifest/ (дата звернення: 20.12.2025).

46. Google. PageSpeed Insights [Електронний ресурс]. – Режим доступу: https://pagespeed.web.dev/ (дата звернення: 20.12.2025).

ДОДАТКИ

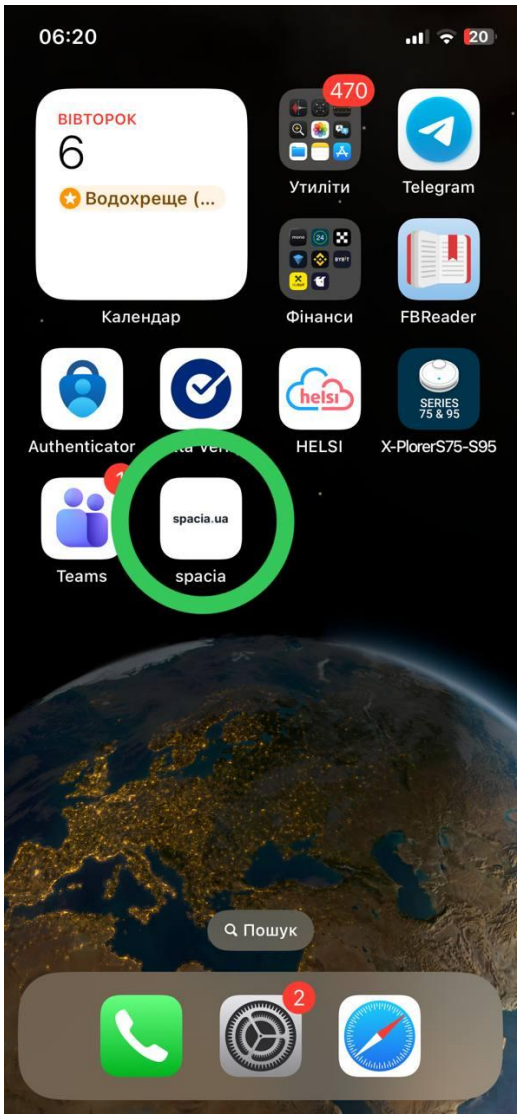
Додаток А

Скріншот режиму AR (з мобільного пристрою)



Додаток Б

“Відображення PWA додатку на мобільному пристрої”



Додаток В

“Лістинг програмного коду основних модулів системи”

Файл ‘app/layout.js’ (Кореневий макет):

```
"use client";

import { Inter } from 'next/font/google';
import "./globals.css";
import { CartProvider } from "../context/CartContext";
import Footer from '../components/Footer';

const inter = Inter({
  subsets: ['latin', 'cyrillic'],
  variable: '--font-inter',
});

export default function RootLayout({ children }) {
  return (
    <html lang="uk" className={`${inter.variable} font-sans`}>
      <head>
        <title>spacia.ua</title>
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <link rel="manifest" href="/manifest.json" />
        <meta name="apple-mobile-web-app-capable" content="yes" />
      </head>

      <CartProvider>
        <body className="flex flex-col min-h-screen">

          <div className="flex-grow">
```

```
          {children}
        </div>

        <Footer />

        <script src="//code.tidio.co/txssd0zgjsqnoyz0lrzkg0w5utbdipc9.js"
async></script>
       </body>
     </CartProvider>
   </html>
  );
}
```

Файл ‘components/ARViewer.js’ (Логіка AR):

```
"use client";

import { useState, useEffect } from 'react';

const MODEL_VIEWER_SCRIPT_ID = "model-viewer-script";
const MODEL_VIEWER_SCRIPT_SRC =
"https://ajax.googleapis.com/ajax/libs/model-viewer/3.5.0/model-viewer.min.js";

export default function ARViewer({ modelSrc }) {
  const [isClient, setIsClient] = useState(false);

  useEffect(() => {
    if (!document.getElementById(MODEL_VIEWER_SCRIPT_ID)) {
      const script = document.createElement('script');
      script.id = MODEL_VIEWER_SCRIPT_ID;
      script.type = "module";
```

```
    script.src = MODEL_VIEWER_SCRIPT_SRC;
    document.body.appendChild(script);
   }
   setIsClient(true);
  }, []);

  return (
   <div className="w-full h-96 border p-4 bg-gray-100 rounded-lg">
    <h3 className="text-xl font-bold text-center mb-4">Демонстрація AR-примірки</h3>

    {isClient ? (
     <model-viewer
       src={modelSrc}
       alt="3D model"
       ar
       ar-modes="webxr scene-viewer quick-look"
       camera-controls
       shadow-intensity="1"
       auto-rotate
       style={{width: '100%', height: '300px'}}
     >
     </model-viewer>
    ) : (
     <p className="w-full h-[300px] flex items-center justify-center text-gray-500">
       Завантаження 3D-моделі...
     </p>
    )}
   </div>
```

```
  );
}
```

Файл ‘app/item/[productId]/page.js’ (динамічна сторінка товару):

```
import { getProductById } from '../../../lib/data';
import { getProductImages } from '../../actions';
import ProductClient from '../../../components/ProductClient';

export default async function ProductPage({ params }) {
  const { productId } = await params;

  const product = getProductById(productId);

  const images = await getProductImages(productId);

  return (
    <ProductClient product={product} initialImages={images} />
  );
}
```

Файл ‘components/ProductCard.js’ (картка товару):

```
import Link from 'next/link';

export default function ProductCard({ product }) {
    return (
        <Link
            href={`/item/${product.id}`}
            className="group bg-white rounded-xl shadow-md overflow-hidden flex
flex-col transition-all duration-300 hover:shadow-2xl hover:-translate-y-2"
        >
```

```
        <div className="relative h-56 w-full bg-white flex items-center justify-center overflow-hidden">
          <img
            src={product.mainImage}
            alt={product.title}
            className="h-full w-full object-contain p-4 transition-transform duration-300 group-hover:scale-110"
          />
        </div>

        <div className="p-5 flex flex-col flex-grow">
          <p className="text-xs font-semibold text-blue-600 uppercase tracking-wide mb-2">
            {product.category}
          </p>
          <h3 className="text-lg font-bold text-gray-900 mb-2 line-clamp-2 group-hover:text-blue-600 transition-colors">
            {product.title}
          </h3>
          <div className="mt-auto pt-4 flex items-center justify-between">
            <p className="text-2xl font-bold bg-gradient-to-r from-blue-600 to-purple-600 bg-clip-text text-transparent">
              {product.price.toFixed(2)} ₴
            </p>
            <span className="text-blue-600 group-hover:translate-x-1 transition-transform">
              →
            </span>
          </div>
        </div>
```

```
      </Link>
    );
}
```

Файл ‘app/cart/page.js’ (сторінка кошику):

```
"use client";

import { useCart } from '../../context/CartContext';
import Header from '../../components/Header';
import { useState } from 'react';
import Link from 'next/link';

export default function CartPage() {
  const { items, removeFromCart, updateQuantity, getTotalPrice } = useCart();
  const [_, setSelectedCategory] = useState("Всі");

  return (
    <>
      <Header onSelectCategory={setSelectedCategory} />

      <main className="container mx-auto px-4 sm:px-6 py-12">
        <h1 className="text-2xl sm:text-3xl font-bold text-gray-900 mb-8">Ваш кошик</h1>

        {items.length === 0 ? (
          <p className="text-gray-600">Ваш кошик порожній. <Link href="/" className="text-blue-600 hover:underline font-medium">Повернутися до магазину</Link></p>
        ) : (
          <div className="grid grid-cols-1 lg:grid-cols-3 gap-8 lg:gap-12">
```

```
          <div className="lg:col-span-2 flex flex-col gap-6">
            {items.map(item => (
              <div key={item.id} className="flex gap-4 bg-white p-4 rounded-lg
shadow-sm border items-center">

                <img
                  src={item.images ? item.images[0] : "/placeholder.png"}
                  alt={item.title}
                  className="w-24 h-24 object-contain rounded"
                />

                <div className="flex-grow">
                  <h2 className="text-base sm:text-lg font-semibold text-gray-
800">{item.title}</h2>
                  <p className="text-gray-500 text-sm capitalize">{item.category}</p>
                  <div className="flex items-center gap-3 mt-2">
                    <div className="flex items-center border border-gray-300 rounded-
lg">
                      <button
                        onClick={() => updateQuantity(item.id, item.quantity - 1)}
                        className="px-3 py-1 text-gray-600 hover:bg-gray-100 rounded-l-
lg"
                      >
                        -
                      </button>
                      <span className="px-2 font-medium text-gray-900 min-w-[30px]
text-center">{item.quantity}</span>
                      <button
                        onClick={() => updateQuantity(item.id, item.quantity + 1)}
```

```
                    className="px-3 py-1 text-gray-600 hover:bg-gray-100 rounded-r-lg"
                  >
                    +
                  </button>
                </div>
                <p className="text-lg font-bold text-gray-900 ml-2">{(item.price * item.quantity).toFixed(2)} ₴</p>
              </div>
            </div>
            <button
              onClick={() => removeFromCart(item.id)}
              className="text-gray-400 hover:text-red-500"
            >
              <svg className="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24" xmlns="http://www.w3.org/2000/svg"><path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M19 7l-.867 12.142A2 2 0 0116.138 21H7.862a2 2 0 01-1.995-1.858L5 7m5 4v6m4-6v6m1-10V4a1 1 0 00-1-1h-4a1 1 0 00-1 1v3M4 7h16"></path></svg>
            </button>
          </div>
        ))}
      </div>

      {/* Checkout Section */}
      <div className="bg-white p-6 rounded-lg shadow-sm border h-fit">
        <h2 className="text-xl sm:text-2xl font-semibold mb-4 text-gray-900">Разом до оплати</h2>
        {(() => {
          const subtotal = parseFloat(getTotalPrice());
```

```
        const shippingCost = subtotal >= 15000 ? 0 : 500;
        const total = subtotal + shippingCost;

        return (
          <>
            <div className="flex justify-between text-base sm:text-lg mb-2 text-gray-700">
              <span>Проміжна сума:</span>
              <span className="font-semibold">{subtotal.toFixed(2)} ₴</span>
            </div>
            <div className="flex justify-between text-base sm:text-lg mb-4 text-gray-700">
              <span>Доставка:</span>
              <span className={shippingCost === 0 ? "text-green-600 font-semibold" : "font-semibold"}>
                {shippingCost === 0 ? "Безкоштовно" : `${shippingCost.toFixed(2)} ₴`}
              </span>
            </div>

            {shippingCost > 0 && (
              <p className="text-sm text-blue-600 mb-4 bg-blue-50 p-2 rounded">
                Додайте товарів ще на {(15000 - subtotal).toFixed(2)} ₴ для безкоштовної доставки!
              </p>
            )}

            <hr className="my-4" />
            <div className="flex justify-between text-xl sm:text-2xl font-bold text-gray-900 mb-6">
```

```
                <span>Всього:</span>
                <span>{total.toFixed(2)} ₴</span>
              </div>
            </>
          );
        })()}

        <button
          className="w-full bg-blue-600 text-white p-3 rounded-lg text-lg font-bold hover:bg-blue-700 shadow-lg hover:shadow-blue-300 transition-all"
          disabled
        >
          Перейти до оплати
        </button>

        <Link
          href="/products"
          className="mt-6 block text-center text-sm text-gray-400 hover:text-blue-600 underline decoration-dotted transition-colors"
        >
          + обрати додатковий товар
        </Link>
      </div>
    </div>
  )}
 </main>
 </>
);
```

```
}
```

Файл ‘components/CartView.js’ (інтерфейс кошику):

```
"use client";

import { useCart } from '../context/CartContext';
import Header from './Header'; // Відносний шлях
import { useState } from 'react';
import Link from 'next/link';

export default function CartView({ categories }) {
  const { items, removeFromCart, getTotalPrice } = useCart();

  const [_, setSelectedCategory] = useState("Всі");

  return (
    <>
      <Header
        categories={categories}
        onSelectCategory={setSelectedCategory}
      />

      <main className="container mx-auto px-6 py-12">
        <h1 className="text-3xl font-bold text-gray-800 mb-8">Ваш кошик</h1>

        {items.length === 0 ? (
          <p className="text-gray-600">Ваш кошик порожній. <Link href="/"
className="text-blue-500 hover:underline">Повернутися до магазину</Link></p>
        ) : (
          <div className="grid grid-cols-1 lg:grid-cols-3 gap-12">
```

```
        <div className="lg:col-span-2 flex flex-col gap-6">
          {items.map(item => (
            <div key={item.id} className="flex gap-4 bg-white p-4 rounded-lg shadow-md items-center">
              <img src={item.image} alt={item.title} className="w-24 h-24 object-cover rounded" />
              <div className="flex-grow">
                <h2 className="text-lg font-semibold">{item.title}</h2>
                <p className="text-gray-500 text-sm">{item.category}</p>
                <p className="text-lg font-bold text-blue-600 mt-1">${item.price}</p>
              </div>
              <button
                onClick={() => removeFromCart(item.id)}
                className="text-red-500 hover:text-red-700"
              >
                Видалити
              </button>
            </div>
          ))}
        </div>

        <div className="bg-white p-6 rounded-lg shadow-md h-fit">
          <h2 className="text-2xl font-semibold mb-4">Разом до оплати</h2>
          <div className="flex justify-between text-lg mb-2">
            <span>Проміжна сума:</span>
            <span className="font-bold">${getTotalPrice()}</span>
          </div>
          <div className="flex justify-between text-lg mb-4">
            <span>Доставка:</span>
```

```
              <span className="font-bold">Безкоштовно</span>
            </div>
            <hr className="my-4" />
            <div className="flex justify-between text-2xl font-bold mb-6">
              <span>Всього:</span>
              <span>${getTotalPrice()}</span>
            </div>

            <button
              className="w-full bg-green-500 text-white p-3 rounded-lg text-lg font-bold opacity-50 cursor-not-allowed"
              disabled
            >
              Перейти до оплати (WIP)
            </button>
          </div>
        </div>
      )}
    </main>
  </>
 );
}
```

Файл ‘context/CartContext.js’ (логіка кошику):

```
"use client";

import { createContext, useState, useContext } from 'react';

const CartContext = createContext();
```

```
export function CartProvider({ children }) {
  const [items, setItems] = useState([]);

  const addToCart = (product) => {
    setItems(prevItems => {
      const existingItem = prevItems.find(item => item.id === product.id);
      if (existingItem) {
        return prevItems.map(item =>
          item.id === product.id ? { ...item, quantity: existingItem.quantity + 1 } : item
        );
      }
      return [...prevItems, { ...product, quantity: 1 }];
    });
  };

  const removeFromCart = (productId) => {
    setItems(prevItems => prevItems.filter(item => item.id !== productId));
  };

  const updateQuantity = (productId, newQuantity) => {
    if (newQuantity < 1) return;

    setItems(prevItems => prevItems.map(item =>
      item.id === productId ? { ...item, quantity: newQuantity } : item
    ));
  };

  const getTotalPrice = () => {
    return items.reduce((total, item) => total + item.price * item.quantity,
0).toFixed(2);
```

```
  };

  return (
    <CartContext.Provider value={{
      items,
      addToCart,
      removeFromCart,
      updateQuantity,
      getTotalPrice,
      itemCount: items.reduce((total, item) => total + item.quantity, 0)
    }}>
      {children}
    </CartContext.Provider>
  );
}

export function useCart() {
  return useContext(CartContext);
}
```