ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально–науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістр

на тему: **«Розроблення вебсервісу для управління каталогом товарів з авторизацією та системою фільтрації на основі React та MySQL»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи та технології
спеціальності 126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_2024
Турбай Євгеній Олександрович
Керівник: Поночовний Юрій Леонідович
Рецензент: Муравльов Володимир Вячеславович

Полтава – 2025 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально–науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«08» листопада 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Турбая Євгенія Олександровича

1. Тема кваліфікаційної роботи:
«Розроблення вебсервісу для управління каталогом товарів з авторизацією та системою фільтрації на основі React та MySQL»,
Керівник роботи: д. т. н., професор, професор кафедри інформаційних систем та технологій Поночовний Юрій Леонідович.
Затверджено наказом закладу вищої освіти від «31» жовтня 2025 року № 1332-ст

2. Строк подання здобувачем вищої освіти роботи «09» грудня 2025 р.

3. Вихідні дані до роботи: наукові джерела та публікації, технічна документація React та MySQL, дані інтернет-ресурсів, середовище виконання Node.js, інструментальні засоби розробки.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1. Аналіз предметної області та визначення вимог.
Розділ 2. Проєктування та обґрунтування вибору технологій.
Розділ 3. Практична реалізація та економічне обґрунтування.

5. Перелік графічного матеріалу: схеми архітектури системи, діаграми UML, схема бази даних, скріншоти інтерфейсу вебсервісу.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Калініченко О. В., к. е. н., доцент, доцент кафедри економіки та публічного управління	24.11.2025	04.12.2025

7. Дата видачі завдання «08» листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1.	Вибір і затвердження теми роботи	29.10.2024 р.	
2.	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	30.10.2024 р. – 08.11.2024 р.	
3.	Опрацювання джерел інформації	11.11.2024 р. – 27.12.2024 р.	
4.	Збір, вивчення і обробка інформації, необхідної для виконання роботи	30.12.2024 р.– 19.01.2025 р.	
5.	Виконання теоретико-методологічного розділу роботи	17.02.2025 р.– 16.05.2025 р.	
6.	Виконання дослідницько-аналітичного розділу роботи	02.06.2025 р.– 13.07.2025 р.	
7.	Виконання проєктно-рекомендаційного розділу роботи	08.09.2025 р.– 14.11.2025 р.	
8.	Оцінювання економічної ефективності результатів дослідження	24.11.2025 р.– 04.12.2025 р.	
9.	Оформлення тексту роботи	05.12.2025 р.– 08.12.2025 р.	
10.	Попередній захист роботи на кафедрі	09.12.2025 р.	
11.	Доопрацювання роботи з урахуванням зауважень і пропозицій	10.12.2025 р.- 14.12.2025 р.	
12.	Нормоконтроль	15.12.2025 р. – 16.12.2025 р.	
13.	Захист кваліфікаційної роботи	18.12.2025 р.	

Здобувач вищої освіти **Євгеній ТУРБАЙ**

Керівник роботи **Юрій ПОНОЧОВНИЙ**

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФУЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

ТУРБАЙ ЄВГЕНІЙ ОЛЕКСАНДРОВИЧ

«РОЗРОБЛЕННЯ ВЕБСЕРВІСУ ДЛЯ УПРАВЛІННЯ КАТАЛОГОМ ТОВАРІВ З АВТОРИЗАЦІЄЮ ТА СИСТЕМОЮ ФІЛЬТРАЦІЇ НА ОСНОВІ REACT ТА MYSQL»

Освітньо-професійна програма
Інформаційні управляючі системи та технології
Спеціальність 126 Інформаційні системи та технології
Ступінь вищої освіти Магістр

РЕФЕРАТ
кваліфікаційної роботи на здобуття кваліфікації –
магістр з інформаційних систем та технологій

Полтава – 2025 року

Кваліфікаційна робота складається зі вступу, 3 розділів, висновків, списку використаних джерел (43 найменувань), 3 додатків. Кваліфікаційна робота містить 7 таблиць, 10 рисунків, викладена на 57 сторінках.

Основний зміст роботи

У першому розділі детально проаналізовано сучасний стан ринку електронної комерції та виявлено ключові проблеми існуючих рішень для онлайн-торгівлі. Встановлено, що традиційні CMS-системи (WordPress, OpenCart), побудовані за архітектурою MPA, часто не відповідають сучасним вимогам щодо швидкодії інтерфейсу та гнучкості налаштування фільтрів через необхідність перезавантаження сторінок . Обґрунтовано доцільність розробки спеціалізованого вебсервісу на основі архітектури SPA (Single Page Application), що дозволяє забезпечити миттєвий відгук системи та покращити користувацький досвід. Сформульовано комплекс функціональних (каталог, кошик, авторизація) та нефункціональних вимог (швидкість завантаження до 1,5 с, безпека даних), а також проведено моделювання варіантів використання системи (UML Use Case) для ролей «Гість» та «Адміністратор» .

У другому розділі обґрунтовано вибір технологічного стеку та спроєктовано архітектуру системи. Для реалізації обрано модифікований стек MERN, де замість MongoDB використано реляційну СУБД MySQL, що забезпечує сувору типізацію даних та підтримку ACID-транзакцій, критично важливих для комерції . Спроєктовано нормалізовану структуру бази даних (3НФ), яка включає сутності товарів, категорій, користувачів та замовлень, із використанням індексів для оптимізації пошукових запитів. Розроблено специфікацію інтерфейсу RESTful API, що базується на принципах stateless-взаємодії та використовує формат JSON для обміну даними між клієнтом та сервером, а також визначено маршрути для CRUD-операцій та аутентифікації .

У третьому розділі висвітлено процес практичної реалізації серверної та клієнтської частин вебсервісу. Розроблено бекенд на Node.js/Express із використанням пулу з'єднань до БД та middleware для логування і захисту (CORS). Реалізовано механізм безпечної авторизації на основі стандарту JWT та хешування паролів (bcrypt) . Клієнтську частину побудовано на бібліотеці React із застосуванням Context API для управління станом та компонентного підходу. Ключовим досягненням є реалізація алгоритму динамічної фільтрації товарів, де SQL-запити формуються на льоту відповідно до обраних параметрів, що значно знижує навантаження на мережу. Проведено комплексне тестування (Postman, Unit-тести) та економічне обґрунтування, яке підтвердило високу рентабельність проєкту (ROI ≈ 255%) та термін окупності близько 5 місяців

Висновки

1. Виконано аналіз сучасного стану ринку електронної комерції та існуючих технологічних рішень для організації онлайн-торгівлі. Виявлено суттєві недоліки традиційних CMS-систем (WordPress, OpenCart), що базуються на архітектурі MPA, зокрема низьку швидкість завантаження сторінок та обмежені можливості інтерактивної фільтрації. Встановлено необхідність переходу до архітектури односторінкових застосунків (SPA) для забезпечення миттєвого відгуку інтерфейсу та покращення користувацького досвіду.

2. Обґрунтовано вибір технологічного стеку MERN, адаптованого під використання реляційної СУБД MySQL, що гарантує цілісність даних та підтримку транзакцій. Спроєктовано архітектуру системи, включаючи нормалізовану базу даних (3НФ) з індексацією ключових полів та специфікацію RESTful API для обміну даними у форматі JSON. Моделювання варіантів використання системи виконано за допомогою UML-діаграм, що дозволило чітко розмежувати права доступу для ролей «Гість» та «Адміністратор» .

3. Розроблено серверну частину вебсервісу на платформі Node.js з використанням фреймворку Express. Реалізовано ефективну взаємодію з базою даних через пул з'єднань, що забезпечує високу продуктивність при обробці паралельних запитів. Впроваджено механізми безпеки, включаючи валідацію вхідних даних, захист від CORS-атак та хешування паролів користувачів за допомогою бібліотеки bcrypt .

4. Проєктовано та реалізовано клієнтську частину на бібліотеці React із використанням компонентного підходу та Context API для управління глобальним станом (кошик, авторизація). Ключовим досягненням є реалізація алгоритму динамічної фільтрації товарів, де SQL-запити формуються на сервері в реальному часі на основі URL-параметрів, а також системи захищеної авторизації на базі стандарту JWT (JSON Web Tokens) .

5. Проведено економічне обґрунтування розробки системи: розрахункова собівартість створення власного програмного продукту становить близько 25 300 грн. Доведено високу інвестиційну привабливість проєкту з показником рентабельності (ROI) на рівні 255% та терміном окупності менше 5 місяців, що підтверджує доцільність відмови від дорогих платних платформ .

6. Проведений аналіз та комплексне тестування (модульне, інтеграційне через Postman) підтвердили працездатність розробленого сервісу, його відповідність вимогам щодо швидкодії (завантаження контенту до 1,5 с) та адаптивності інтерфейсу для мобільних пристроїв.

Таким чином, поставлені задачі розв’язано у повному обсязі. Напрямком подальших досліджень є впровадження технології Server-Side Rendering (SSR) для покращення SEO-оптимізації, інтеграція з платіжними системами (LiqPay, Stripe) та розвиток проєкту до повноцінного Progressive Web App (PWA) з можливістю офлайн-роботи.

Список публікацій здобувача

1. Турбай Є. Переваги використання SPA-архітектури при розробці вебкаталогів товарів. *Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару*, 25 листопада 2025 р. Полтава: ПДАУ, 2025. С. 105–106.

АНОТАЦІЯ

Турбай Є. О. «Розроблення вебсервісу для управління каталогом товарів з авторизацією та системою фільтрації на основі React та MySQL». Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття ступеня вищої освіти магістр за освітньо-професійною програмою Інформаційні управляючі системи та технології спеціальності 126 Інформаційні системи та технології. Полтавський державний аграрний університет, Полтава, 2025.

Виконано аналіз сучасних підходів до веб-розробки в сфері електронної комерції та обґрунтовано переваги використання архітектури SPA. Спроєктовано та розроблено вебсервіс для продажу комп'ютерної техніки з використанням стеку технологій MERN (адаптованого під MySQL). Реалізовано серверну частину (REST API) для обробки запитів та клієнтську частину на React з механізмами динамічної фільтрації товарів, захищеною авторизацією (JWT) та управлінням кошиком. Проведено тестування та економічне обґрунтування розробленої системи.

Ключові слова: електронна комерція, SPA, React, Node.js, MySQL, REST API, JWT, фільтрація даних, вебсервіс.

ANNOTATION

Turbai Ye. O. «Development of a Web Service for Managing a Product Catalog with Authorization and a Filtering System Based on React and MySQL». Qualification work as a manuscript.

Qualification work for obtaining the Master's degree in the educational-professional program Information Management Systems and Technologies, specialty 126 Information Systems and Technologies. Poltava State Agrarian University, Poltava, 2025.

An analysis of modern web development approaches in e-commerce has been conducted, and the advantages of using SPA architecture have been substantiated. A web service for selling computer hardware has been designed and developed using the MERN technology stack (adapted for MySQL). The server-side (REST API) for request processing and the client-side on React with dynamic product filtering mechanisms, secure authorization (JWT), and shopping cart management have been implemented. Testing and an economic justification of the developed system have been performed.

Keywords: e-commerce, SPA, React, Node.js, MySQL, REST API, JWT, data filtering, web service.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Інтерфейс програмування застосунків

CMS – Система керування контентом

CSS – Каскадні таблиці стилів

E-commerce – Електронна комерція

HTML – Мова гіпертекстової розмітки

HTTP – Протокол передачі гіпертексту

HTTPS – Захищений протокол передачі гіпертексту

IT – Інформаційні технології

JS – Мова програмування JavaScript

JWT – Веб-токен JSON

MERN – Стек технологій

MPA – Багатосторінковий застосунок

MySQL – Система керування реляційними базами даних

PHP – Мова програмування гіпертекстових препроцесорів

REST – Архітектурний стиль для розподілених систем

SEO – Пошукова оптимізація

SPA – Односторінковий застосунок

SQL – Мова структурованих запитів

СУБД – Система управління базами даних

СЦ – Сервісний центр

ТЗ – Технічне завдання

UI – Інтерфейс користувача

UX – Досвід користувача

ВСТУП

Актуальність теми. Сучасний етап розвитку світової економіки характеризується стрімкою цифровізацією всіх сфер бізнесу. Ринок електронної комерції (e–commerce) демонструє стабільне зростання, поступово витісняючи традиційні форми торгівлі. Особливо це стосується сегменту комп'ютерної техніки та електроніки, де споживачі віддають перевагу онлайн–покупкам через можливість швидкого порівняння характеристик та цін [1, 2].

Важливим аспектом функціонування сучасних інтернет-магазинів є дотримання нормативно-правової бази, що регулює дистанційну торгівлю. Розробка та впровадження інформаційного сервісу здійснювалися з урахуванням вимог чинного законодавства, зокрема Закону України «Про електронну комерцію». Цей закон визначає організаційно-правові засади діяльності у сфері електронної комерції, встановлює порядок вчинення електронних правочинів та гарантує захист прав усіх учасників цифрового ринку [3].

В умовах жорсткої конкуренції ключовими факторами успіху інтернет–магазину стають не лише асортимент та ціна, а й якість програмної реалізації сервісу: швидкість завантаження, зручність інтерфейсу (UI/UX), адаптивність під мобільні пристрої та безпека даних. Традиційні підходи до веброзробки (MPA – Multi Page Application) часто не можуть забезпечити миттєвий відгук інтерфейсу, якого очікують сучасні користувачі. Тому актуальним є завдання розробки інформаційних сервісів на основі архітектури SPA (Single Page Application) з використанням сучасних JavaScript–фреймворків, що дозволяє створити швидкий, масштабований та зручний інструмент для підтримки продажів.

Метою кваліфікаційної роботи є проєктування та програмна реалізація інформаційного сервісу підтримки продажу комп'ютерної техніки, який

забезпечує ефективну взаємодію з клієнтами, автоматизацію обробки замовлень та високу швидкодію роботи.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз предметної області та існуючих аналогів на ринку електронної комерції;

– обґрунтувати вибір стеку технологій (React, Node.js, MySQL) для реалізації проєкту;

– спроєктувати архітектуру бази даних та розробити специфікацію RESTful API;

– виконати програмну реалізацію серверної частини (Backend) для обробки запитів та роботи з даними;

– розробити клієнтську частину (Frontend) з використанням компонентного підходу та адаптивної верстки;

– реалізувати механізми авторизації (JWT), фільтрації товарів та роботи з кошиком;

– провести тестування розробленого сервісу та виконати економічне обґрунтування доцільності його впровадження.

Об’єктом дослідження є процес автоматизації продажів комп'ютерної техніки в мережі Інтернет.

Предметом дослідження є методи та засоби створення високонавантажених вебсервісів на основі архітектури клієнт – сервер з використанням JavaScript технологій.

Методи дослідження. У роботі використано методи системного аналізу (для дослідження предметної області та аналогів), метод моделювання (для проєктування бази даних та архітектури ПЗ), об’єктно-орієнтованого та функціонального програмування (для написання коду), а також методи економічного аналізу (для розрахунку окупності проєкту).

Наукова новизна одержаних результатів полягає у подальшому розвитку підходів до побудови SPA-застосунків для електронної комерції

шляхом інтеграції бібліотеки React із реляційною базою даних MySQL через REST API, що дозволило поєднати гнучкість інтерфейсу з надійністю зберігання структурованих даних.

Практичне значення одержаних результатів. Розроблений вебсервіс «CompMax» є повністю функціональним програмним продуктом, готовим до впровадження у діяльність торгівельних підприємств. Він дозволяє зменшити витрати на обслуговування клієнтів, пришвидшити обробку замовлень та покращити користувацький досвід завдяки миттєвому відгуку інтерфейсу. Економічні розрахунки підтверджують, що термін окупності проєкту становить близько 5 місяців.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на наукових конференціях, семінарах.

Публікації. За результатами проведеного дослідження опубліковано тези: «Переваги використання SPA архітектури при розробці вебкаталогів товарів», Студентські роботи за науковою тематикою кафедри інформаційних систем та технологій: матеріали XXII щорічного міждисциплінарного семінару, 25 листопада 2025 р. Полтава: ПДАУ, 2025 р.

Структура та обсяг кваліфікаційної роботи логічно пов’язані з задачами досліджень. Робота містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел, додатки. Загальний обсяг текстової частини дипломної роботи складає 57 сторінок формату А4. Вона містить 10 рисунків і 7 таблиць. В роботі використано 43 науково-технічних джерел.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИЗНАЧЕННЯ ВИМОГ

1.1 Аналіз предметної області

Сучасний етап розвитку світової економіки характеризується глобальною цифровізацією бізнес–процесів. Електронна комерція (e–commerce) перетворилася з додаткового каналу продажів на основний інструмент взаємодії між продавцем та покупцем. Згідно з аналітичними даними, частка онлайн–торгівлі щорічно зростає, що зумовлює підвищення вимог до програмного забезпечення, яке обслуговує цю сферу.

Предметною областю даної кваліфікаційної роботи є технології створення веборієнтованих інформаційних систем для роздрібної торгівлі, зокрема спеціалізованих інтернет–магазинів комп'ютерної техніки. Специфіка цієї ніші полягає у великій кількості технічних характеристик товарів (тактова частота процесора, обсяг пам'яті, тип матриці тощо), що вимагає від програмного продукту просунутих можливостей структурування даних та їх фільтрації.

Ключовим елементом будь-якої торгової платформи є електронний каталог – систематизований перелік товарних позицій, доступний користувачам через вебінтерфейс. Аналіз предметної області дозволяє виділити основні проблеми, притаманні існуючим рішенням застарілого зразка:

Особливу увагу в сучасній електронній комерції слід приділяти концепції Omnichannel, яка передбачає безшовну інтеграцію різних каналів комунікації з клієнтом. Хоча в рамках даної роботи розглядається виключно веб-інтерфейс, проєктування системи повинно враховувати потенційну можливість розширення, наприклад, створення мобільного додатку, який використовуватиме ту ж саму базу даних та API.

Варто також зазначити, що згідно з останніми дослідженнями поведінкової психології покупців, критичним фактором прийняття рішення про покупку є час завантаження контенту (Time to Interactive). Дослідження Google показують, що 53% користувачів залишають мобільний сайт, якщо він завантажується довше 3 секунд. Це накладає жорсткі обмеження на вибір архітектурних рішень. Традиційні підходи, де сервер генерує кожну сторінку «з нуля», часто не можуть впоратися з піковими навантаженнями під час маркетингових акцій (Black Friday, Cyber Monday), що призводить до втрати прибутку. Тому перехід до клієнтського рендерингу (Client-Side Rendering) є не просто трендом, а економічно обґрунтованою необхідністю.

Графічна ілюстрація тенденцій зростання обсягів електронної комерції та кореляція цього процесу з підвищенням вимог до продуктивності веб-ресурсів наведена на рис. 1.1.

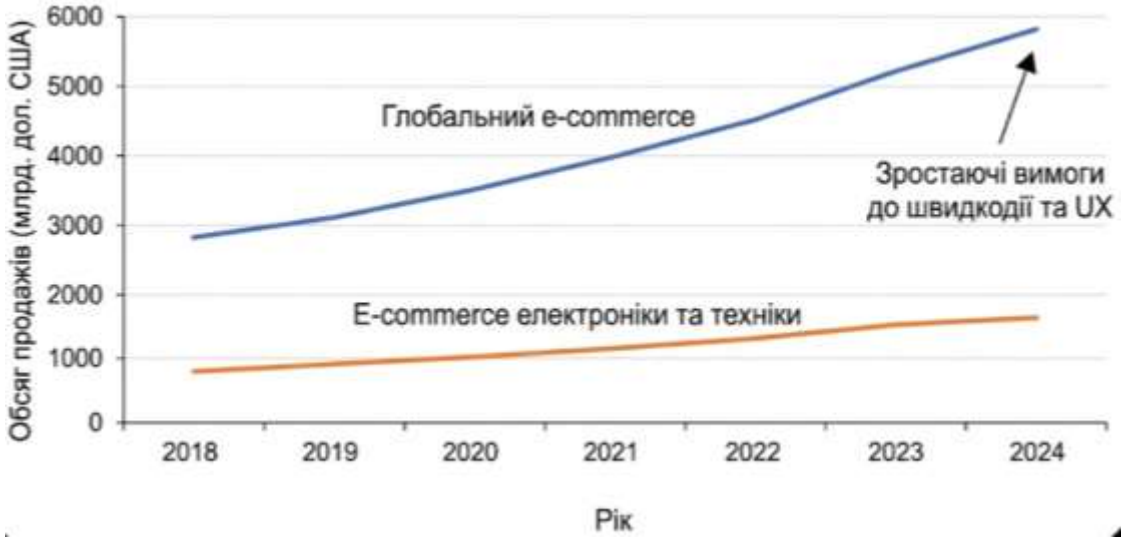

Рисунок 1.1 – Динаміка росту ринку електронної комерції та вимоги до швидкодії вебсервісів

Традиційні вебсайти, побудовані за архітектурою MPA (Multi–Page Application), вимагають повного перезавантаження сторінки при кожній дії користувача (сортування, перехід на іншу сторінку пагінації, застосування

фільтру). Це значно погіршує користувацький досвід (User Experience) та збільшує час пошуку потрібного товару.

При великому асортименті товарів критично важливим стає наявність механізму фасетної фільтрації, який дозволяє відбирати товари за багатьма параметрами одночасно. Ефективне управління каталогом неможливе без розмежування прав доступу. Система повинна надійно захищати адміністративну панель від несанкціонованого доступу, забезпечуючи при цьому зручний інтерфейс для менеджерів.

Для вирішення окреслених проблем сучасна веброзробка зміщує фокус у бік архітектури односторінкових застосунків (SPA – Single Page Application). Використання бібліотек, таких як React, дозволяє реалізувати інтерфейс, який миттєво реагує на дії користувача без звернення до сервера за новою розміткою сторінки. Серверна частина при цьому трансформується у набір API–методів, що повертають лише необхідні дані у форматі JSON [4].

У контексті зберігання даних стандартом де–факто для систем електронної комерції залишаються реляційні бази даних (RDBMS). Специфіка предметної області (наявність чіткої структури товару: категорія, бренд, ціна, характеристики) робить використання реляційної моделі, реалізованої в СУБД MySQL, найбільш доцільним рішенням. Це забезпечує цілісність даних, відсутність дублювання інформації та високу швидкість виконання складних пошукових запитів.

Підсумовуючи результати аналізу предметної області, можна стверджувати, що сфера онлайн-торгівлі комп'ютерною технікою характеризується високим рівнем конкуренції та специфічними вимогами до структуризації даних. Ключовим викликом для сучасних веб-систем у цій ніші стає необхідність обробки великих масивів технічних характеристик товарів без втрати продуктивності інтерфейсу. Встановлено, що традиційні підходи до організації каталогу часто не здатні задовольнити зростаючі очікування користувачів щодо швидкості пошуку та зручності навігації. Це обумовлює актуальність розробки спеціалізованого програмного

забезпечення, яке, на відміну від універсальних рішень, буде спроєктоване з урахуванням специфіки продажу складної електроніки, забезпечуючи баланс між інформативністю та ергономікою [5].

1.2 Огляд існуючих ринкових рішень та сервісів – аналогів

Для обґрунтування доцільності розробки власного вебсервісу було проведено дослідження існуючих на ринку програмних рішень, що дозволяють створювати каталоги товарів. Ринок пропонує широкий вибір інструментів: від універсальних систем управління контентом (CMS) до вузькоспеціалізованих фреймворків. Кожна категорія рішень має свої переваги та обмеження, які впливають на продуктивність, гнучкість налаштування та вартість володіння продуктом.

Для наочного порівняння технічних можливостей існуючих рішень та проєктуємого вебсервісу було складено порівняльну характеристику (таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика систем управління каталогами товарів

Критерій порівняння	WordPress (WooCommerce)	Magento (Adobe Commerce)	Проєктований сервіс (React + MySQL)
Архітектура	MPA (Monolithic)	MPA (Complex Monolith)	SPA (Decoupled API–first)
Швидкість відгуку	Середня	Залежить від серверу	Висока (Virtual DOM)
Гнучкість фільтрації	Обмежена плагінами	Висока, але складна	Повна кастомізація алгоритмів
Навантаження на мережу	Високе (HTML render)	Високе (Heavy payloads)	Мінімальне (JSON exchange)
Вартість розробки	Низька	Висока	Оптимальна

Одним із найпоширеніших рішень у світі є поєднання CMS WordPress із плагіном WooCommerce. Ця платформа залучає розробників низьким порогом входження та наявністю величезної кількості готових шаблонів. Однак аналіз показує, що універсальність системи є її основним недоліком. Велика кількість надлишкового коду та залежність від сторонніх модулів призводять до сповільнення швидкості завантаження сторінок. Крім того, стандартні механізми фільтрації в WooCommerce часто реалізовані через синхронні запити, що вимагає перезавантаження інтерфейсу при кожній зміні критеріїв пошуку [6].

Для великих корпоративних проєктів часто обирають платформу Magento (Adobe Commerce). Це потужне професійне рішення, яке забезпечує високу масштабованість та містить вбудовані інструменти для управління складними атрибутами товарів. ПротеMagento є надзвичайно вимогливою до серверних ресурсів СУБД, а її архітектура є складною для швидкої кастомізації логіки інтерфейсу. Розробка та підтримка систем на базі Magento потребує залучення вузькопрофільних спеціалістів, що робить її економічно недоцільною для середніх проєктів, які потребують високої швидкості роботи клієнтської частини [7].

Третьою категорією є спеціалізовані платформи, такі як OpenCart. Це рішення орієнтоване виключно на електронну комерцію і пропонує більш легковажну архітектуру порівняно з Magento. Проте OpenCart, як і більшість традиційних CMS, базується на архітектурі MPA. Хоча існують розширення для впровадження асинхронної фільтрації (AJAX), вони часто конфліктують із шаблонами дизайну та не забезпечують тієї плавності роботи, яку надає сучасний реактивний підхід на базі бібліотек типу React [8].

Окрім зазначених платформ, варто розглянути рішення на базі конструкторів сайтів (SaaS-платформи), таких як Shopify або Wix. Хоча вони пропонують найшвидший старт, їхнім головним недоліком є повна залежність від вендора («Vendor Lock-in»). Власник магазину не має доступу до вихідного коду, не може оптимізувати структуру бази даних під

специфічні потреби бізнесу та змушений сплачувати щомісячну абонплату, яка зростає пропорційно до обігу.

На противагу цьому, розробка власного (Custom) рішення, яка пропонується в даній роботі, забезпечує повний контроль над інтелектуальною власністю, дає можливість необмеженої масштабованості та інтеграції з будь-якими сторонніми сервісами (CRM, ERP, служби доставки) без необхідності купівлі дорогих ліцензій. Це стратегічна інвестиція, яка окупається в довгостроковій перспективі за рахунок відсутності комісійних платежів стороннім платформам.

Аналіз наведених у таблиці даних свідчить про те, що готові ринкові рішення здебільшого орієнтовані на універсальність, що неминуче призводить до компромісів у продуктивності. Традиційні системи MPA–типу не здатні забезпечити миттєву динамічну фільтрацію без значних затримок на передачу HTML–розмітки. Проєктований у роботі вебсервіс на базі React та MySQL усуває ці обмеження шляхом використання SPA–архітектури, де сервер відповідає лише за передачу корисних даних, а клієнтська логіка забезпечує реактивний відгук інтерфейсу.

Принципова відмінність у механізмі оновлення інтерфейсу між використанням реального DOM та технології Virtual DOM, що лежить в основі React, схематично зображена на рисунку 1.2.

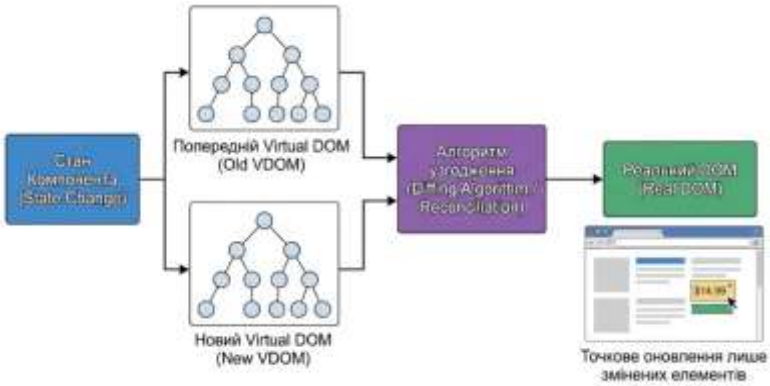

Рисунок 1.2 – Схема роботи алгоритму Virtual DOM та оновлення інтерфейсу в React

Використання кастомної розробки на сучасному стеку дозволяє реалізувати систему фільтрації, яка працює в реальному часі на боці клієнта, що є критично важливою вимогою для сучасних інтернет–магазинів комп'ютерної техніки. Такий підхід забезпечує повний контроль над архітектурою бази даних MySQL, виключає проблему надлишковості функціоналу та дозволяє створити максимально легкий та швидкий програмний продукт, адаптований під конкретні бізнес–завдання.

Узагальнюючи огляд існуючих ринкових інструментів, варто зазначити, що жодна з розглянутих платформ (WordPress, Magento, OpenCart) не є ідеальним рішенням для поставленого завдання без суттєвих доопрацювань. Коробочні CMS-системи, хоч і пропонують швидкий старт, часто страждають від архітектурної надлишковості, що негативно впливає на показники швидкодії та ускладнює масштабування під високі навантаження. З іншого боку, SaaS-конструктори створюють залежність від вендора та обмежують доступ до програмного коду. На основі цього зроблено висновок про доцільність розробки власного вебсервісу (Custom Development), що дозволить реалізувати унікальну бізнес-логіку фільтрації та забезпечити повний контроль над оптимізацією продуктивності, що є критичним конкурентним фактором [9].

1.3 Формулювання функціональних та нефункціональних вимог до вебсервісу

На основі аналізу предметної області та розроблених макетів інтерфейсу було сформульовано комплексні вимоги до вебсервісу, які визначають його поведінку та якісні характеристики. Чітке визначення цих вимог є критичним етапом, що закладає фундамент для подальшого проєктування архітектури системи. Проєктована система повинна забезпечувати повний цикл взаємодії користувача з каталогом товарів: від

пошуку до оформлення замовлення, а також надавати інструменти для адміністрування контенту [36].

Функціональне ядро клієнтської частини системи (Front–office) має базуватися на принципах зручності та інформативності. Головна сторінка повинна містити динамічні елементи залучення уваги, зокрема слайдер з промо–матеріалами («Hero Slider») та блоки трендових товарів, реалізація яких у прототипі спирається на адаптивні скрипти. Ключовою вимогою до каталогу є реалізація механізму багатофакторної фільтрації, що дозволяє користувачам відбирати товари за ціновим діапазоном, брендами та специфічними характеристиками, а також сортувати їх за релевантністю. Детальна сторінка товару повинна надавати вичерпну інформацію про продукт, включаючи інтерактивну галерею зображень та можливість вибору атрибутів, таких як колір або конфігурація.

Окрему увагу слід приділити процесу покупки та авторизації. Система повинна підтримувати повноцінну роботу кошика: додавання товарів, динамічний перерахунок загальної вартості та управління кількістю позицій. Для забезпечення персоналізації та безпеки необхідна реалізація підсистеми реєстрації та автентифікації користувачів. Це дозволить розмежувати права доступу, надавши звичайним відвідувачам доступ до публічної частини, а адміністраторам – до панелі управління (Back–office) для виконання CRUD–операцій над товарами та категоріями.

Окрім функціональних можливостей, система повинна відповідати низці нефункціональних вимог, що гарантують її якість та надійність. Пріоритетною вимогою є висока продуктивність, яка досягається завдяки використанню архітектури SPA (Single Page Application) на базі бібліотеки React. Це забезпечить миттєвий відгук інтерфейсу на дії користувача, зокрема при фільтрації контенту, без перезавантаження сторінки. Інтерфейс вебсервісу має бути повністю адаптивним (Responsive Web Design), коректно відображатися на пристроях будь–якого типу, що реалізується через гнучку верстку та медіа–запити. З точки зору безпеки, критично важливим є захист

персональних даних: паролі користувачів повинні зберігатися у базі даних MySQL виключно у вигляді хеш–сум, а всі вхідні дані мають проходити валідацію для запобігання SQL–ін'єкціям та XSS–атакам [10].

Таким чином, на основі проведеного аналізу сформовано цілісний реєстр вимог до майбутньої системи, який слугуватиме фундаментом для подальшого проєктування. Чітке розмежування функціональних вимог (що система повинна робити) та нефункціональних (як система повинна працювати) дозволяє мінімізувати ризики на етапах розробки та тестування. Визначені пріоритети щодо швидкодії (час завантаження до 1.5 с), безпеки (захист персональних даних) та адаптивності інтерфейсу є обов'язковими критеріями якості. Формалізація цих вимог гарантує, що кінцевий програмний продукт не лише вирішуватиме бізнес-задачі замовника, але й відповідатиме сучасним стандартам якості веб-застосунків [11, 35].

1.4 Моделювання варіантів використання системи

Для деталізації функціональних вимог та визначення меж системи було використано метод моделювання варіантів використання (Use Case Modeling) мовою UML. Цей метод дозволяє формалізувати сценарії взаємодії користувачів із вебсервісом, визначити ролі (акторів) та їхні повноваження в системі [12].

На основі аналізу предметної області та структури розробленого інтерфейсу виділено двох основних акторів:

– гість (Guest) – неавторизований користувач, який має доступ до публічної частини каталогу. Це основна цільова аудиторія сайту;

– адміністратор (Admin) – авторизований користувач (співробітник магазину), який має розширені права доступу для управління контентом.

Візуальне представлення всіх варіантів використання та зв'язків між ними наведено на діаграмі (рис. 1.3).

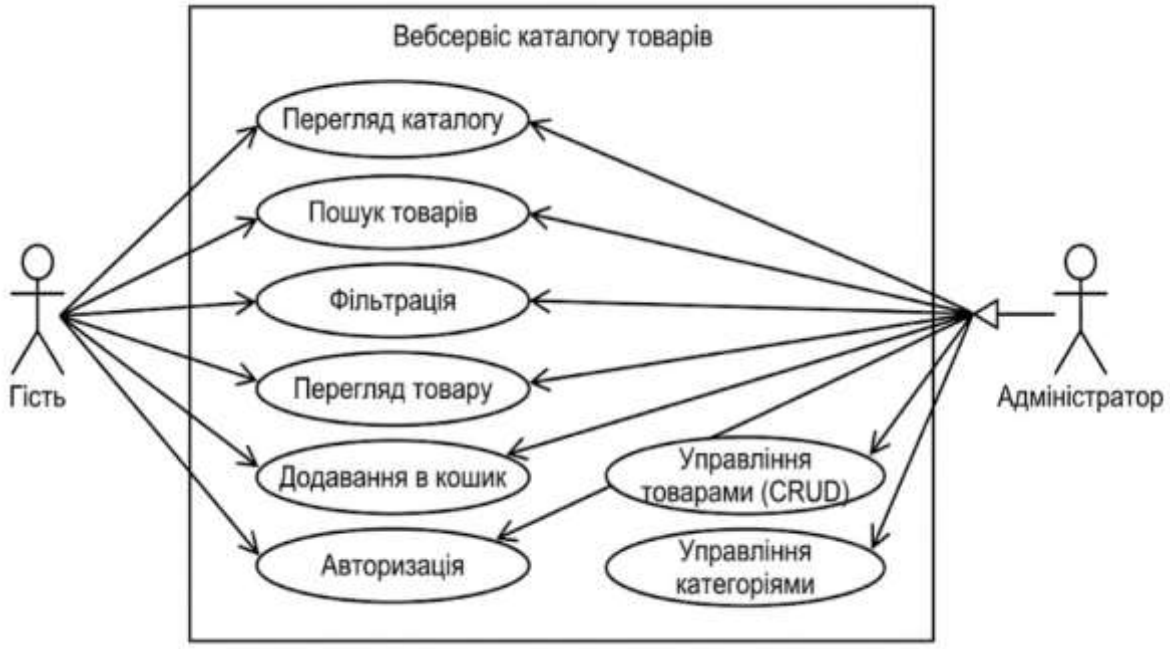

Рисунок 1.3 – Діаграма варіантів використання вебсервісу

Базовий сценарій взаємодії для актора «Гість», починається з входу користувача на головну сторінку, де він має доступ до навігаційного меню та промо–блоків. Основні варіанти використання для цього актора включають:

– перегляд каталогу. Користувач може переглядати списки товарів, розбиті за категоріями;

– пошук та фільтрація. Для швидкого знаходження потрібного товару Гість використовує пошуковий рядок у шапці сайту або панель фільтрів у каталозі (за ціною, брендом тощо);

– перегляд детальної інформації. Перехід на сторінку окремого товару для ознайомлення з характеристиками та фотографіями;

– робота з кошиком. Додавання товарів до кошика та перегляд його вмісту;

– авторизація. Ініціювання процесу входу в систему для отримання прав доступу.

Натомість, адміністратор є спеціалізованим актором, який повинен пройти процедуру аутентифікації. Після успішного входу він отримує доступ до захищеної частини API.

До його унікальних варіантів використання належать:

– управління товарами (CRUD). Створення нових карток товарів, редагування існуючих (зміна ціни, опису) та видалення неактуальних позицій;

– управління категоріями. Створення та редагування структури каталогу.

Представлена на рис. 1.3 діаграма демонструє чітке розмежування функціоналу між ролями. Актор «Адміністратор» через відношення узагальнення (Generalization) успадковує всі можливості «Гостя» (наприклад, пошук та перегляд), але додатково має ексклюзивний доступ до варіантів використання групи «Управління контентом». Така модель лежить в основі проєктованої системи безпеки та розмежування прав доступу на рівні маршрутів серверного API.

Висновки до розділу 1

Проведений аналіз сучасного стану ринку електронної комерції засвідчив стійку тенденцію до зростання вимог користувачів щодо якості та швидкодії веб-ресурсів. Дослідження існуючих підходів до побудови торговельних майданчиків виявило, що традиційні архітектурні рішення часто стають стримуючим фактором для подальшого розвитку бізнесу через обмежені можливості масштабування та недостатню інтерактивність інтерфейсу.

Встановлено, що перехід до сучасних моделей побудови веб-застосунків є необхідною умовою для забезпечення конкурентоспроможності на ринку. Визначення чітких функціональних та нефункціональних вимог до системи дозволило сформувати концептуальне бачення майбутнього програмного продукту, який має поєднувати високу продуктивність, безпеку

даних та зручність користування, що в сукупності забезпечує ефективну взаємодію з цільовою аудиторією.

Також результати порівняльного аналізу існуючих ринкових рішень та CMS-платформ підтвердили доцільність відмови від використання універсальних шаблонних систем. Виявлені архітектурні обмеження традиційних підходів, зокрема щодо швидкості реакції інтерфейсу та гнучкості налаштування фільтрів, стали вагомим аргументом на користь розробки власного веб-сервісу на базі архітектури SPA. Такий підхід дозволить уникнути надлишковості функціоналу та забезпечити повний контроль над оптимізацією процесів обробки даних.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1 Обґрунтування вибору архітектури вебсервісу

Вибір архітектурного патерну є фундаментальним етапом проєктування будь-якої інформаційної системи, оскільки саме він визначає подальшу логіку взаємодії компонентів, можливості масштабування та показники продуктивності програмного продукту. У контексті розробки сучасного інтернет–магазину комп'ютерної техніки було проведено детальний порівняльний аналіз двох домінуючих підходів до побудови вебзастосунків: традиційної багатосторінкової архітектури (Multi–Page Application – MPA) та сучасної односторінкової архітектури (Single–Page Application – SPA).

Традиційна модель MPA, на базі якої функціонує більшість класичних CMS (наприклад, OpenCart або WordPress), базується на тісній зв'язаності клієнтської та серверної частин. Логіка роботи такої системи передбачає, що при кожній дії користувача – будь то перехід у категорію «Відеокарти», застосування фільтра за ціною або додавання товару до кошика – браузер надсилає запит на сервер. Сервер, у свою чергу, виконує обчислення, генерує нову HTML–сторінку повністю і повертає її клієнту. Цей процес неминуче призводить до повного перезавантаження сторінки у вікні браузера.

Аналіз недоліків MPA–архітектури для предметної області електронної комерції виявив такі критичні проблеми:

– низька швидкість реакції інтерфейсу. Користувач змушений чекати перезавантаження всієї сторінки навіть при незначних змінах контенту;

– надлишковий трафік. Сервер щоразу передає повторювані елементи дизайну (шапку, футер, меню), що збільшує навантаження на мережу, особливо для мобільних користувачів;

– складність розділення розробки. Frontend та Backend розробники змушені працювати в одному репозиторії, що ускладнює командну роботу.

На противагу цьому, архітектура SPA, обрана для реалізації даного дипломного проєкту, пропонує принципово іншу модель взаємодії. Її суть полягає в тому, що вебзастосунок завантажує лише один HTML–документ (оболонку) під час першого візиту користувача. Вся подальша взаємодія з сервером відбувається асинхронно (AJAX) у фоновому режимі. Сервер виступає виключно як постачальник даних у форматі JSON, а рендеринг інтерфейсу відбувається безпосередньо в браузері клієнта за допомогою JavaScript.

Для наочної демонстрації відмінностей у принципах обробки запитів між цими архітектурами розроблено порівняльну схему (рис. 2.1).

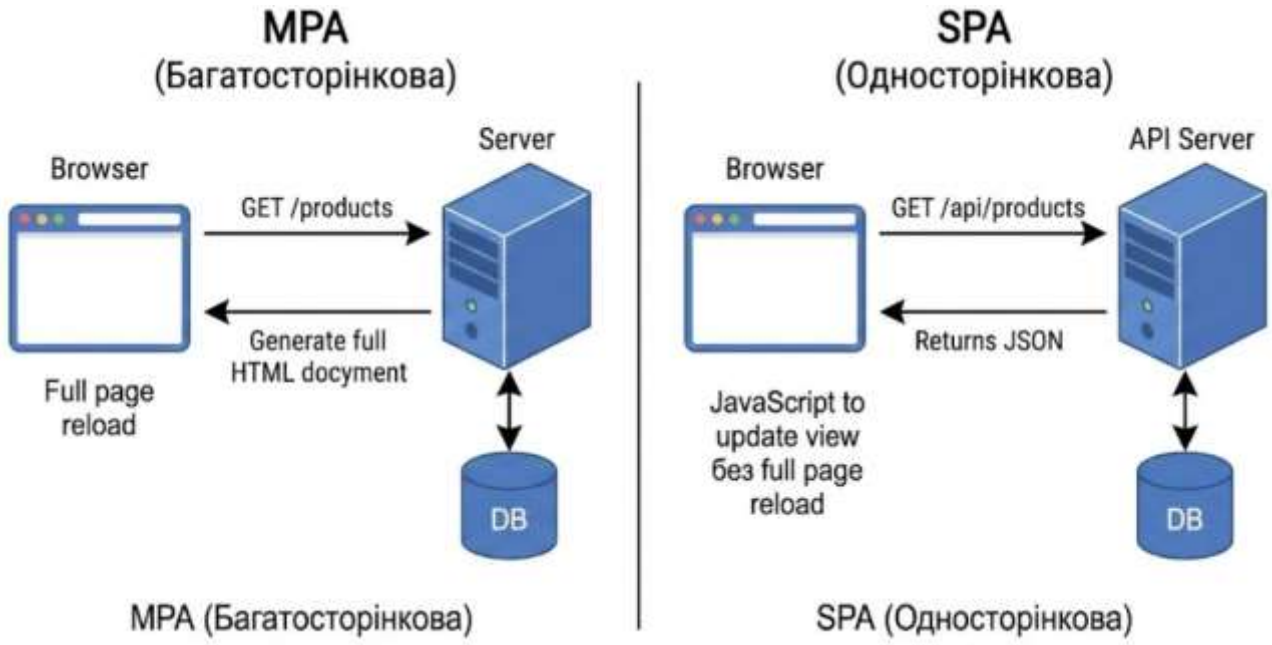

Рисунок 2.1 – Порівняння життєвого циклу запиту в MPA та SPA архітектурах

Як видно з наведеної схеми, у SPA–архітектурі відсутній етап повторної генерації HTML–коду на стороні сервера. Це дозволяє досягти

ефекту миттєвого відгуку інтерфейсу, що є критично важливим для утримання уваги покупця в інтернет–магазині. Крім того, така архітектура дозволяє чітко розмежувати зони відповідальності: серверна частина (Backend) реалізує бізнес–логіку та роботу з БД через API, а клієнтська частина (Frontend) відповідає виключно за відображення та взаємодію з користувачем.

Для остаточного обґрунтування вибору було проведено порівняльний аналіз обох підходів за ключовими критеріями якості програмного забезпечення. Результати аналізу зведено в табл. 2.1.

Таблиця 2.1 – Порівняльна характеристика архітектурних підходів MPA та SPA

Критерій порівняння	Multi–Page Application (MPA)	Single–Page Application (SPA)
Швидкість першого завантаження	Висока (браузер одразу отримує готовий HTML)	Середня (потрібно завантажити JS–бандл)
Швидкість роботи в процесі використання	Низька (постійні перезавантаження сторінок)	Дуже висока (миттєве оновлення контенту)
Навантаження на сервер	Високе (генерація HTML для кожного клієнта)	Низьке (передача тільки JSON–даних)
Обсяг переданих даних	Великий (HTML розмітка + дані)	Мінімальний (тільки корисні дані)
Користувацький досвід (UX)	Переривчастий («миготіння» екрану)	Плавний (native–like application)
Масштабованість	Складна (монолітна структура)	Висока (Front/Back відокремлені)

Важливим аспектом при виборі SPA-архітектури є питання пошукової оптимізації (SEO). Історично склалося так, що пошукові роботи (crawlers) мали труднощі з індексацією сайтів, контент яких генерується динамічно через JavaScript. Однак сучасні алгоритми Google (зокрема Googlebot) вже вміють виконувати JS-код перед індексацією.

Крім того, SPA-архітектура дозволяє реалізувати концепцію PWA (Progressive Web App). Це технологія, яка дозволяє вебсайту працювати як нативний додаток: зберігати дані в кеш, працювати в офлайн-режимі та

відправляти push-сповіщення. Хоча повна реалізація PWA виходить за рамки даної роботи, обрана архітектура на базі React закладає фундамент для легкої трансформації магазину в PWA в майбутньому, що є суттєвою конкурентною перевагою.

Підсумовуючи результати аналізу, для розробки вебсервісу управління каталогом товарів було обрано архітектуру SPA. Це рішення обумовлене необхідністю реалізації складної системи динамічної фільтрації товарів (за ціною, брендом, характеристиками), яка повинна працювати без затримок. Використання SPA дозволяє забезпечити високу конкурентоспроможність сервісу завдяки кращому користувацькому досвіду, а також спрощує подальшу підтримку системи завдяки модульній структурі та використанню REST API [13].

2.2 Обґрунтування вибору технологій стеку

Успішна реалізація вебсервісу значною мірою залежить від правильно підібраного інструментарію. Технологічний стек проєкту формувався на основі вимог до продуктивності, швидкості розробки, масштабованості та підтримки спільноти. Для реалізації архітектури Single Page Application (SPA) було обрано стек MERN (MySQL, Express, React, Node.js), який дозволяє використовувати мову JavaScript на всіх рівнях розробки: від бази даних до інтерфейсу користувача.

Ключовим компонентом клієнтської частини (Frontend) обрано бібліотеку React.js, розроблену компанією Facebook (Meta). Головною перевагою React є концепція віртуального DOM (Virtual DOM). Це легковагова копія реального DOM–дерева, яка зберігається в пам'яті. Коли стан застосунку змінюється (наприклад, користувач вводить текст у поле пошуку або додає товар до кошика), React спочатку вносить зміни у віртуальну структуру, потім за допомогою алгоритму узгодження

(Reconciliation) обчислює мінімально необхідну кількість змін і лише після цього пакетно оновлює реальний DOM браузера. Такий підхід дозволяє уникнути «важких» операцій перемальовування всієї сторінки, забезпечуючи високу продуктивність інтерфейсу [14, 15].

Для обґрунтування вибору React було проведено порівняльний аналіз із найближчими конкурентами на ринку – фреймворками Angular та Vue.js (таблиця 2.2).

Таблиця 2.2 – Порівняльний аналіз технологій для розробки клієнтського інтерфейсу

Критерій порівняння	React (Обрано)	Angular	Vue.js
Тип архітектури	Бібліотека (View Library)	Повноцінний MVC–фреймворк	Прогресивний фреймворк
Механізм рендерингу	Virtual DOM (ефективне оновлення)	Incremental DOM	Virtual DOM
Мова розробки	JavaScript / JSX	TypeScript (обов'язково)	JavaScript / HTML–шаблони
Крива навчання	Середня (гнучкість підходів)	Висока (велика кількість концепцій)	Низька (простий синтаксис)
Екосистема та спільнота	Найбільша у світі, величезна кількість готових пакетів (NPM)	Велика, орієнтована на Enterprise–сектор	Швидкозростаюча, але менша за React
Гнучкість	Висока (дозволяє обирати бібліотеки для роутингу, стану)	Низька (жорстка структура проєкту)	Середня

Аналіз даних таблиці показує, що React забезпечує найкращий баланс між продуктивністю та гнучкістю розробки, що є критичним для створення динамічного каталогу товарів.

Для реалізації серверної частини (Backend) обрано платформу Node.js. Це середовище виконання JavaScript, побудоване на рушії V8 від Google Chrome. Основною перевагою Node.js є неблокуюча, подієво–орієнтована модель вводу–виводу (Event–driven, Non–blocking I/O). Це означає, що сервер не створює окремий потік для кожного підключення, а обробляє тисячі одночасних з'єднань в одному потоці, використовуючи механізм черги подій

(Event Loop). Така архітектура ідеально підходить для вебсервісів електронної комерції, де потрібно обробляти велику кількість коротких запитів (отримання списку товарів, фільтрація, авторизація) без складних математичних обчислень на сервері [16].

В якості системи управління базами даних (СУБД) обрано MySQL. Оскільки предметна область інтернет–магазину оперує чітко структурованими даними (товари, категорії, замовлення, користувачі) зі сталими зв'язками, використання реляційної моделі є найбільш доцільним. MySQL забезпечує повну відповідність принципам ACID (Atomicity, Consistency, Isolation, Durability), що гарантує надійність транзакцій – наприклад, при списанні товару зі складу в момент оформлення замовлення. Крім того, MySQL є безкоштовним рішенням з відкритим кодом, що знижує вартість впровадження системи [17, 18].

Вибір реляційної моделі даних (SQL) на противагу нереляційній (NoSQL, наприклад MongoDB) обумовлений природою даних електронної комерції. Транзакції в інтернет-магазині вимагають суворого дотримання принципів ACID (Atomicity, Consistency, Isolation, Durability):

– атомарність. Операція списання коштів та створення замовлення має виконуватися як єдине ціле. Якщо одна частина не вдалася, скасовується все;

– узгодженість. Дані повинні відповідати всім правилам (наприклад, не можна створити замовлення на товар, якого не існує);

– ізольованість. Паралельні транзакції не повинні впливати одна на одну;

– довговічність. Підтверджені дані не можуть бути втрачені навіть при збої живлення. MySQL забезпечує ці властивості «з коробки», тоді як NoSQL рішення часто жертвують узгодженістю заради швидкості запису, що є неприпустимим для фінансових операцій.

Обґрунтування технологічного стеку MERN (MySQL, Express, React, Node.js) підтверджує його відповідність поставленим завданням.

Використання JavaScript як єдиної мови програмування для клієнтської та серверної частин дозволяє уніфікувати процес розробки та спростити підтримку коду. Бібліотека React забезпечує високу продуктивність рендерингу завдяки Virtual DOM, а платформа Node.js гарантує ефективну обробку великої кількості одночасних I/O запитів, що є типовим для магазинів. Вибір реляційної СУБД MySQL, на противагу NoSQL рішенням, забезпечує необхідну строгість структури даних та транзакційну надійність, що є критичним для фінансових операцій та складських обліків.

2.3 Проєктування структури бази даних

Проєктування бази даних є критично важливим етапом розробки, оскільки від якості спроєктованої схеми залежить швидкість обробки пошукових запитів, цілісність даних та можливість подальшого масштабування системи. Виходячи з обраного технологічного стеку, для зберігання даних використовується реляційна СУБД MySQL.

На етапі концептуального моделювання, базуючись на бізнес–процесах інтернет–магазину та структурі інтерфейсу, було виділено ключові сутності предметної області. Основні об'єкти системи включають:

- користувачі (Users). Суб'єкти системи, які проходять авторизацію;
- товари (Products). Об'єкти продажу з набором характеристик;
- категорії (Categories). Логічні групи для класифікації товарів;
- замовлення (Orders). Інформація про здійснені покупки;
- елементи замовлення (Order Items). Проміжна сутність для зв'язку товарів із замовленнями.

Для забезпечення ефективності зберігання даних було проведено нормалізацію бази до третьої нормальної форми (3НФ). Це дозволило

уникнути дублювання інформації (наприклад, назва категорії не зберігається в таблиці кожного товару, а винесена в окремий довідник) [19].

Центральним елементом бази даних є таблиця products. Її структура розроблена таким чином, щоб забезпечити швидке відображення карток товарів на фронтенді. Детальний опис атрибутів цієї таблиці наведено в таблиці 2.3.

Таблиця 2.3 – Структура таблиці products (Товари)

Назва поля	Тип даних	Опис та призначення
id	INT (PK)	Унікальний ідентифікатор товару (Auto Increment).
category_id	INT (FK)	Зовнішній ключ для зв'язку з таблицею категорій.
title	VARCHAR(255)	Повна назва товару (наприклад, «GIGABYTE GeForce RTX4070»).
price	DECIMAL(10,2)	Поточна ціна товару. Використовується для сортування та розрахунків.
old_price	DECIMAL(10,2)	Стара ціна (для відображення знижки, як реалізовано у верстці).
image_url	VARCHAR(255)	Шлях до зображення файлу (наприклад, assets/products/cpu–1.png).
stock_qty	INT	Кількість товару на складі.
is_featured	BOOLEAN	Флаг для виведення у блок «Рекомендовані».
created_at	TIMESTAMP	Дата додавання товару (для сортування «Новинки»).

Для реалізації підсистеми безпеки спроєктовано таблицю users. Важливою особливістю є те, що паролі користувачів не зберігаються у відкритому вигляді. Для цього використовується поле password_hash, яке містить зашифрований рядок.

Встановлення зв'язків (Relationships) Між таблицями встановлено реляційні зв'язки, що забезпечують цілісність даних:

– categories – Products (1:M). Одна категорія (наприклад, «Відеокарти») може містити багато товарів. При видаленні категорії передбачено правило SET NULL, щоб товари не зникали з бази, а ставали «без категорії»;

– users – Orders (1:M). Один користувач може мати історію з багатьох замовлень;

– orders – Products (M:M). Оскільки одне замовлення містить багато товарів, а один товар може бути в багатьох замовленнях, цей зв'язок реалізовано через проміжну таблицю order_items. Вона містить поля order_id, product_id та quantity (кількість конкретного товару в чеку).

Для візуалізації повної структури бази даних та зв'язків між сутностями розроблено ER–діаграму (Entity–Relationship Diagram), яка представлена на рисунку 2.2.

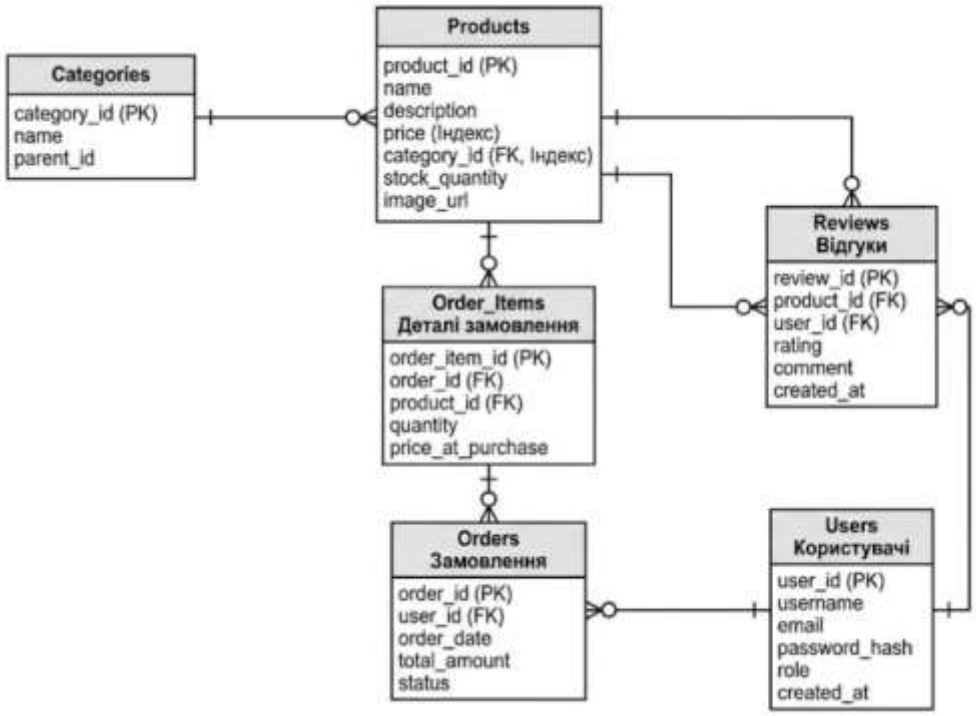

Рисунок 2.2 – ER–діаграма бази даних вебсервісу

Описана структура бази даних повністю покриває функціональні вимоги вебсервісу. Використання індексів для полів price та category_id дозволяє оптимізувати виконання SQL–запитів при фільтрації, що є критичним для забезпечення швидкодії SPA–застосунку [20, 37].

Спроєктована схема бази даних, приведена до третьої нормальної форми, забезпечує надійне зберігання інформації без надлишкового

дублювання. Розроблена структура таблиць та зв'язків між ними повністю покриває потреби предметної області, дозволяючи зберігати ієрархічні категорії, параметричні характеристики товарів та історію замовлень користувачів. Наявність правильно налаштованих зовнішніх ключів та індексів гарантує цілісність даних (Referential Integrity) та високу швидкість виконання пошукових SQL-запитів, що є фундаментом для швидкої роботи всього вебсервісу під навантаженням.

2.4 Проєктування інтерфейсу RESTful API

Оскільки архітектура системи базується на принципі Single Page Application (SPA), де клієнтська частина повністю відокремлена від серверної, критично важливим етапом проєктування є розробка уніфікованого інтерфейсу обміну даними. Для забезпечення взаємодії між фронтендом на React та бекендом на Node.js було спроєктовано прикладний програмний інтерфейс (API), побудований за архітектурним стилем REST (Representational State Transfer).

Вибір REST–архітектури обумовлений її масштабованістю, простотою інтеграції та використанням стандартних методів протоколу HTTP. У спроєктованій системі кожна сутність (товар, категорія, замовлення) розглядається як ресурс, що має унікальний ідентифікатор (URI). Маніпуляції над ресурсами здійснюються за допомогою стандартних HTTP–дієслів:

– get. Отримання представлення ресурсу (наприклад, завантаження списку товарів для каталогу). Цей метод є безпечним та ідемпотентним, тобто його багаторазовий виклик не змінює стан сервера;

– post. Створення нового ресурсу (наприклад, реєстрація користувача або оформлення замовлення);

– put/patch. Оновлення існуючого ресурсу (наприклад, зміна ціни товару адміністратором);

– delete. Видалення ресурсу з системи.

Обмін даними відбувається у форматі JSON (JavaScript Object Notation). Цей формат було обрано замість XML через його компактність та нативну сумісність із мовою JavaScript, що дозволяє уникнути додаткових витрат обчислювальних ресурсів на стороні клієнта при парсингу відповідей сервера [21].

Принцип обробки запиту в такій архітектурі зображено на діаграмі послідовності (рисунок 2.3).

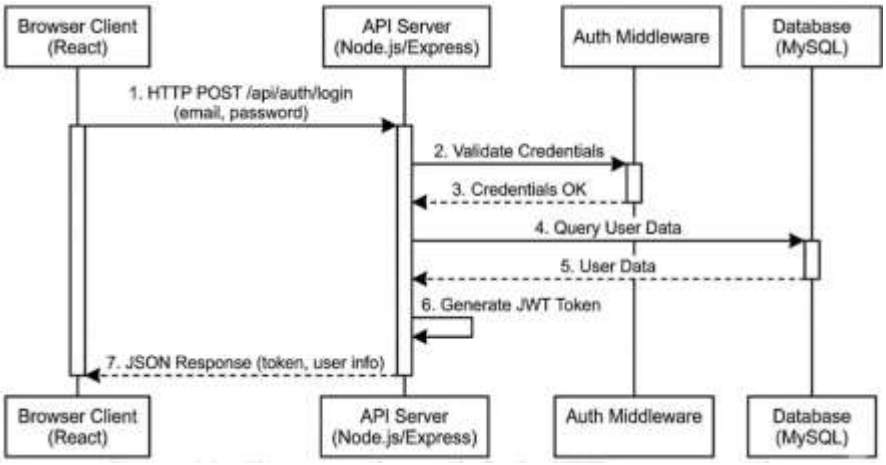

Рисунок 2.3 – Діаграма послідовності обробки HTTP–запиту в системі

Як видно з діаграми, процес взаємодії ініціюється на стороні клієнта (Client). Компонент React відправляє асинхронний запит (використовуючи Fetch API або бібліотеку Axios) до відповідного маршруту сервера. Контролер API (API Controller) приймає запит, виконує валідацію вхідних даних та звертається до моделі даних (Data Model) для виконання операцій з базою даних MySQL. Після отримання результату сервер формує відповідь у форматі JSON разом із відповідним HTTP–кодом статусу (наприклад, 200 OK для успішного запиту або 404 Not Found, якщо ресурс не знайдено).

Важливою характеристикою спроєктованого API є відсутність збереження стану клієнта на сервері (Statelessness). Це означає, що сервер не зберігає інформацію про сесію користувача між запитами. Натомість, кожен запит до захищених маршрутів (наприклад, «Мій кошик» або «Адмін–панель») повинен містити токен авторизації (JWT – JSON Web Token) у заголовку Authorization. Такий підхід значно спрощує масштабування системи, оскільки будь–який сервер у кластері може обробити запит користувача, не звертаючись до спільного сховища сесій.

На основі функціональних вимог до системи, описаних у першому розділі (фільтрація, пошук, кошик), було розроблено специфікацію маршрутів API. Маршрути логічно згруповано за контролерами, що відповідають основним сутностям предметної області (таблиця 2.4).

Таблиця 2.4 – Специфікація маршрутів RESTful API

Ресурс (Сутність)	Метод	Маршрут (End-point)	Опис функціоналу	Параметри запиту (Query/Body)
Auth (Авторизація)	POST	/api/auth/register	Реєстрація	Body: { email, password, name }
	POST	/api/auth/login	Вхід у систему	Body: { email, password }
Products (Товари)	GET	/api/products	Отримання списку товарів	Query: ?page=1&category=gpu&min_price=1000&sort=price_asc
	GET	/api/products/:id	Отримання повної інформації.	Param: id
	POST	/api/products	Додавання нового товару	Body: { title, price, category_id, ... }
	PUT	/api/products/:id	Редагування даних	Body: { price, stock_qty, ... }
	DELETE	/api/products/:id	Видалення товару	Param: id
Categories (Категорії)	GET	/api/categories	Отримання дерева категорій	–

Особливу увагу при проєктуванні було приділено маршруту GET /api/products. Оскільки однією з ключових вимог є швидка фільтрація каталогу, цей маршрут підтримує гнучкий набір параметрів запиту (Query Parameters). Наприклад, запит /api/products?category=gpu&price_max=20000 автоматично трансформується контролером у відповідний SQL–запит WHERE до бази даних. Це дозволяє завантажувати на клієнт лише ті дані, які відповідають критеріям пошуку користувача, зменшуючи навантаження на мережу та прискорюючи відображення сторінки.

Розроблена специфікація API є основою для подальшої програмної реалізації системи та забезпечує чіткий контракт взаємодії між фронтенд та бекенд частинами проєкту.

Висновки до розділу 2

Розроблена специфікація програмного інтерфейсу (API) визначає чіткий контракт взаємодії між клієнтською та серверною частинами системи. Дотримання принципів REST-архітектури та використання стандартних методів HTTP забезпечує передбачуваність поведінки системи та легкість її інтеграції з іншими сервісами. Спроєктована система маршрутизації та форматів обміну даними (JSON) дозволяє ефективно передавати інформацію про товари, користувачів та замовлення, мінімізуючи обсяг трафіку. Крім того, stateless-природа спроєктованого API спрощує потенційне горизонтальне масштабування серверної частини в майбутньому.

У процесі проєктування системи було обґрунтовано вибір архітектурних та технологічних рішень, які відповідають сучасним стандартам веб-розробки. Аналіз переваг та недоліків різних підходів дозволив визначити оптимальну конфігурацію технологічного стеку, що забезпечує баланс між гнучкістю розробки та стабільністю функціонування

програмного комплексу. Важливим аспектом стало проєктування структури даних, яка гарантує цілісність інформації та оптимізацію процесів її обробки.

Розроблена специфікація програмних інтерфейсів заклала надійний фундамент для побудови масштабованої системи з чітким розмежуванням зон відповідальності між клієнтською та серверною частинами. Такий підхід створює передумови для легкого розширення функціоналу в майбутньому та спрощує процес інтеграції з іншими сервісами, що є критично важливим для довгострокової експлуатації інформаційної системи.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

3.1 Розробка серверної частини (backend) та бази даних

Практична реалізація вебсервісу розпочалася зі створення серверної інфраструктури, яка слугує фундаментом для обробки даних та бізнес–логіки застосунку. Відповідно до спроєктованої у другому розділі архітектури, бекенд реалізовано на платформі Node.js з використанням реляційної СУБД MySQL. Головним завданням цього етапу було створення фізичної моделі бази даних, наповнення її первинними даними, взятими з макетів верстки, та розробка REST API для взаємодії з клієнтською частиною.

Фізична реалізація бази даних розпочалася зі створення схеми compmax_db на сервері MySQL. На основі логічної моделі було розроблено SQL–скрипти для створення таблиць (DDL – Data Definition Language). Таблиця products містить поля для зберігання назви, поточної ціни (price), старої ціни для відображення знижок (old_price), а також шляху до зображення у файловій системі (image_url), що відповідає атрибутам src тегів <img> у наданих HTML–файлах.

Лістинг SQL–коду для створення ключових таблиць системи наведено нижче:

```
CREATE TABLE categories (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   slug VARCHAR(100) NOT NULL UNIQUE –
);
CREATE TABLE products (
   id INT AUTO_INCREMENT PRIMARY KEY,
   category_id INT,
   title VARCHAR(255) NOT NULL,
```

```
    price DECIMAL(10, 2) NOT NULL,
    old_price DECIMAL(10, 2) DEFAULT NULL,
    image_url VARCHAR(255) NOT NULL,
    stock_qty INT DEFAULT 0,
    is_trending BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(id) ON DELETE SET
NULL);
```

Після створення структури було виконано наповнення бази тестовими даними (Data Seeding). Інформація про товари (назви, ціни, зображення) була екстракьована з файлу index.html та інших сторінок прототипу і імпортована до таблиць categories та products за допомогою SQL–запитів INSERT. Це дозволило одразу отримати робочу базу даних з реалістичним контентом для подальшої розробки та тестування.

Наступним кроком стала розробка серверного застосунку на Node.js. Було ініціалізовано проєкт та встановлено необхідні залежності: фреймворк express для маршрутизації запитів, драйвер mysql2 для роботи з базою даних та пакет cors для дозволу крос–доменних запитів із фронтенду. Для забезпечення безпеки та гнучкості налаштування параметри підключення до бази даних (хост, користувач, пароль) було винесено у змінні оточення (файл .env).

Реалізацію взаємодії з базою даних виконано з використанням пулу з'єднань (Connection Pool), що підвищує продуктивність сервера при обробці паралельних запитів. Приклад модуля конфігур// db.js – Модуль підключення до MySQL:

```
const mysql = require('mysql2');
require('dotenv').config();
const pool = mysql.createPool({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
```

```
    database: process.env.DB_NAME,
    password: process.env.DB_PASSWORD,
    waitForConnections: true,
    connectionLimit: 10, // Максимальна кількість одночасних з'єднань
    queueLimit: 0
});
module.exports = pool.promise();
```

На базі цього підключення було реалізовано контролери REST API. Головний контролер товарів (productController.js) містить методи для обробки GET–запитів. Наприклад, метод отримання списку всіх товарів виконує SQL–запит SELECT до бази даних, перетворює отриманий результат у формат JSON і відправляє його клієнту. Цей ендпоінт замінює собою статичний масив даних, який міг би використовуватися у файлі script.js на етапі прототипування.

Важливим елементом розробки серверної частини є реалізація механізму проміжного програмного забезпечення (Middleware). У Express.js middleware – це функції, які мають доступ до об'єкта запиту (req), об'єкта відповіді (res) та наступної функції у циклі запит-відповідь.

У розробленому сервісі middleware використовується для вирішення кількох задач:

– логування. Фіксація інформації про кожен вхідний запит (IP-адреса, час, метод) для подальшого аналізу та аудиту безпеки;

– обробка помилок (Error Handling). Централізований перехоплювач помилок дозволяє уникати падіння сервера при виникненні виключних ситуацій та повертати клієнту зрозумілі повідомлення про помилки у форматі JSON, приховуючи при цьому деталі реалізації стеку, що є важливою вимогою безпеки;

– cors (Cross-Origin Resource Sharing). Налаштування заголовків безпеки, які дозволяють браузеру виконувати запити до API з іншого домену (домену клієнтської частини).

Реалізація серверної частини на платформі Node.js створила надійний фундамент для функціонування всього вебсервісу. Розроблена модульна структура бекенду, що включає контролери, моделі даних та проміжне програмне забезпечення (middleware), забезпечує чистоту коду та зручність його супроводу. Впровадження механізмів безпеки, таких як валідація вхідних даних та захист від CORS-атак, гарантує стійкість системи до зовнішніх загроз. Використання пулу з'єднань з базою даних дозволило досягти високої продуктивності при обробці запитів, що підтверджує правильність архітектурних рішень, прийнятих на етапі проєктування [22, 23].

3.2 Розробка клієнтської частини (Frontend) на React

Розробка клієнтського інтерфейсу полягала у трансформації попередньо створеного статичного прототипу (HTML/CSS/JS) у динамічний односторінковий застосунок (SPA). Головною метою цього етапу було забезпечення модульності коду та реактивності інтерфейсу, що дозволяє користувачеві взаємодіяти з каталогом без перезавантаження сторінки.

Для створення базової структури проєкту було використано інструмент збірки Vite, який забезпечує значно вищу швидкість компіляції порівняно з традиційним Webpack. Ініціалізацію виконано командою npm create vite@latest client — —template react [24, 38].

Архітектура папок проєкту була організована за компонентним принципом, що дозволяє логічно згрупувати елементи інтерфейсу:

– src/assets – директорія для статичних ресурсів. Сюди було перенесено зображення з папки assets прототипу (банери, фото товарів, логотипи брендів). Також сюди інтегровано глобальний файл стилів style.css, який містить базові налаштування типографіки (var(—system–ui)), кольорову палітру та медіа–запити.

– src/components – для багаторазових компонентів (шапка, картка товару).

– src/pages – для компонентів–сторінок (Головна, Каталог, Кошик).

– src/context – для управління глобальним станом застосунку (Context API).

На основі аналізу файлу index.html було проведено декомпозицію монолітної верстки на незалежні React–компоненти. Це дозволило усунути дублювання коду.

До ключових реалізованих компонентів можна віднести:

– header. Компонент створено на основі тегу <header> з файлу index.html. Логіка мобільного меню, яка в прототипі реалізовувалася через функції copyMenu та обробники подій у script.js, була переписана з використанням хука useState. Тепер відкриття/закриття меню керується зміною стану змінної isMenuOpen, що автоматично оновлює DOM.

– heroslider. Блок головного слайдера ( ) був винесений в окремий модуль. Для збереження функціоналу свайпів та автопрокрутки, реалізованого в оригіналі через бібліотеку Swiper, у React–проєкт було інтегровано пакет swiper/react. Це дозволило використовувати слайдер як декларативний компонент [31].

– productcard. Це один із найважливіших компонентів системи. Він відповідає за відображення картки товару у списках. Верстка компонента базується на блоці , взятому з файлу index.html. Компонент приймає об'єкт product через пропси та динамічно підставляє дані:

– зображення. product.image_url;

– знижка. розраховується автоматично, якщо є product.old_price;

– ціна. product.price.

Лістинг адаптованого компонента ProductCard.jsx:

```
import { Link } from 'react–router–dom';
const ProductCard = ({ product }) => {
```

```
 // Розрахунок відсотка знижки
 const discount = product.old_price
  ? Math.round(((product.old_price – product.price) / product.old_price) * 100)
  : null;
 return (
  <div className=«item»>
   <div className=«media»>
    <div className=«thumbnail object–cover»>
     <Link to={`/product/${product.id}`}>
      <img src={product.image_url} alt={product.title} />
     </Link>
    </div>
    <div className=«hoverable»>
     <ul>
      <li className=«active»><a href=«#»><i className=«ri–heart–
line»></i></a></li>
      <li><Link to={`/product/${product.id}`}><i className=«ri–eye–
line»></i></Link></li>
     </ul>
    </div>
    {discount && <div className=«discount circle flexcenter»><span>–
{discount}%</span></div>}
   </div>
   <div className=«content»>
    <h3 className=«main–links»>
      <Link to={`/product/${product.id}`}>{product.title}</Link>
    </h3>
    <div className=«rating»>
     <div className=«stars»></div>
     <span className=«mini–text»>({product.reviews_count} відгуків)</span>
    </div>
```

```
<div className=«price»>
  <span className=«current»>{product.price} грн</span>
  {product.old_price && <span className=«normal mini–
text»>{product.old_price} грн</span>}
</div>
</div>
```

Для навігації між сторінками без перезавантаження браузера впроваджено бібліотеку React Router DOM. У файлі App.jsx налаштовано маршрути, які відповідають структурі HTML–файлів прототипу [28].

Таблиця 3.1 – Карта маршрутів клієнтського застосунку

URL–шлях	Компонент (Page)	Відповідний файл прототипу	Опис
/	HomePage	index.html	Головна сторінка з банерами та трендами.
/catalog	CatalogPage	page–category.html	Сторінка категорії з фільтрами та сіткою товарів.
/product/:id	ProductPage	page–single.html	Динамічна сторінка товару. Параметр :id використовується для завантаження даних конкретного продукту.
/cart	CartPage	cart.html	Сторінка кошика з таблицею обраних товарів.
/checkout	CheckoutPage	page–offer.html	Сторінка оформлення замовлення.

Особливу увагу було приділено управлінню станом кошика. В оригінальному прототипі логіка додавання товару реалізовувалася через прямі маніпуляції з DOM у файлі addCartClicked.js (пошук елементів за класом .cart–trigger та зміна innerHTML лічильника).

У React–версії такий підхід є неприпустимим. Тому було створено контекст CartContext, який надає глобальний доступ до масиву обраних товарів та методів addToCart, removeFromCart:

– при натисканні кнопки «Додати в кошик» викликається функція контексту, яка оновлює стан;

– компонент шапки (Header) автоматично перерендеюється, відображаючи актуальну кількість товарів на іконці кошика div.item–floating.

Також було адаптовано функціонал вкладок (Tabs), який використовується на сторінці товару (page–single.html) для перемикання між описом та відгуками. Логіка з script.js (функція showPage та додавання класу .active) була замінена на локальний стан компонента: const [activeTab, setActiveTab] = useState('description').

Для наочності архітектури клієнтської частини розроблено діаграму дерева компонентів (рисунок 3.1).

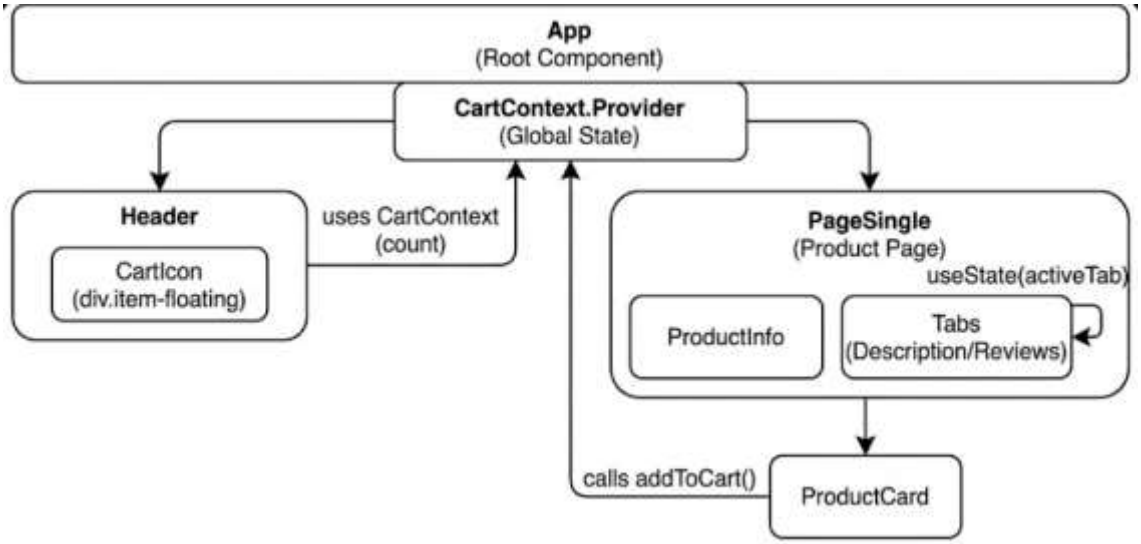

Рисунок 3.1 – Дерево компонентів React–застосунку

Варто детальніше зупинитися на проблемі передачі даних між компонентами, відомій як «Prop Drilling». У класичному React дані передаються від батьківського компонента до дочірнього через властивості (props). У глибокій ієрархії компонентів це призводить до того, що проміжні компоненти змушені передавати дані, які вони самі не використовують, лише для того, щоб доставити їх на нижні рівні.

Для вирішення цієї проблеми у проєкті було застосовано Context API. Це вбудований механізм React, який дозволяє «телепортувати» дані (наприклад, стан кошика або статус авторизації користувача) безпосередньо

до будь-якого компонента в дереві, оминаючи проміжні ланки. Це значно спрощує код, робить його більш читабельним та полегшує рефакторинг у майбутньому. Використання хука useContext дозволяє підписатися на зміни контексту лише в тих компонентах, де це дійсно необхідно, що оптимізує продуктивність рендерингу.

Розробка клієнтського інтерфейсу з використанням бібліотеки React дозволила створити сучасний, інтерактивний та зручний для користувача фронтенд. Компонентний підхід забезпечив високий рівень повторного використання коду (DRY), що значно пришвидшило процес розробки та спростило тестування окремих елементів інтерфейсу. Використання інструменту збірки Vite та сучасних можливостей CSS дозволило оптимізувати розмір кінцевого бандлу, що позитивно вплинуло на швидкість завантаження сторінок. Реалізована структура проєкту є гнучкою та дозволяє легко додавати нові функціональні модулі без необхідності переробки існуючого коду [25].

3.3 Реалізація ключового функціоналу: авторизація та фільтрація

Після налаштування базової маршрутизації та відображення контенту наступним етапом стала імплементація складної бізнес–логіки. Ключовими вимогами до системи є забезпечення розмежування прав доступу між звичайними відвідувачами та адміністратором, а також реалізація механізму пошуку товарів серед великого асортименту.

У вихідному прототипі інтерфейсу (index.html) функціонал акаунту був представлений лише статичною іконкою та випадаючим меню. Для перетворення цього на робочу систему безпеки було обрано стратегію на основі токенів JWT (JSON Web Tokens). Цей підхід дозволяє серверу залишатися «stateless» (не зберігати сесії в пам'яті), що підвищує швидкодію.

Серверна реалізація (Backend): На стороні Node.js було створено контролер авторизації authController.js. Він містить два ключові методи:

– реєстрація. Приймає email та пароль. Критично важливим аспектом є безпека, тому паролі не зберігаються у відкритому вигляді. Використано бібліотеку bcrypt для хешування паролів перед записом у базу даних MySQL;

– вхід (Login). Перевіряє наявність користувача в базі та порівнює хеш введеного пароля зі збереженим. У разі успіху генерується підписаний JWT–токен, який містить userID та role (наприклад, 'admin') [41].

У React–застосунку створено контекст AuthContext, який зберігає стан поточного користувача. При успішному вході отриманий токен записується у localStorage браузера, що дозволяє користувачеві залишатися в системі навіть після перезавантаження сторінки.

Для захисту адміністративних розділів створено компонент ProtectedRoute. Він перевіряє роль користувача перед рендерингом сторінки. Якщо звичайний користувач спробує зайти в панель адміністрування, система автоматично перенаправить його на головну сторінку.

Фрагмент програмного коду, що демонструє реалізацію методу відправки даних форми авторизації на сервер та обробку отриманої відповіді на стороні клієнта, наведено на рисунку 3.2.

```
export AuthForm = () => {
  const [email, [password]] = useState();
  const [password, password]] = useState();

  const handleSubmit = async (c) => {
    try {
      fetch('post', 'POST /api/auth/login', email { {paswords, password }.password } }));
    } catch (e) {
    subsmit.log();
  };

  return (
    <form handleSubmit=AuthsForms ">
      <input type={email} emaile=emailtime={email, email} />
      <input type={password, password} lassit={password, &state=password} />
      <button name="submit">, submit</button>
    </form>
  );
}
```

Рисунок 3.2 – Фрагмент коду компонента авторизації на React

Однією з головних переваг розробленого SPA є миттєва фільтрація товарів. У статичній верстці (page–category.html) блок фільтрів (div.filter–block) містив лише HTML–розмітку для категорій, брендів та цінового повзунка. У програмній реалізації ці елементи стали інтерактивними контролерами, що керують станом списку товарів.

Логіка роботи фільтрів:

– стан фільтрів. У компоненті каталогу створено об'єкт стану filters, який зберігає поточні значення: обрані бренди, діапазон цін (min, max) та параметри сортування;

– синхронізація з URL. При зміні будь–якого фільтра (наприклад, вибір бренду «Asus») спрацьовує хук useEffect. Він формує рядок запиту (Query String), наприклад: ?brand=Asus&price_max=15000, і відправляє асинхронний запит fetch до API.

Обробка на сервері: Бекенд динамічно конструює SQL–запит до бази даних, додаючи умови WHERE відповідно до отриманих параметрів. Це дозволяє фільтрувати дані на рівні СУБД, що значно ефективніше, ніж фільтрація масиву на клієнті [29].

Нижче наведено приклад реалізації логіки формування запиту на React:

```
useEffect(() => {
   const fetchProducts = async () => {
      // Перетворюємо об'єкт фільтрів у рядок параметрів URL
      const queryParams = new URLSearchParams(filters).toString();
      const response = await
fetch(`http://localhost:5000/api/products?${queryParams}`);
      const data = await response.json();
      setProducts(data);
   };

   fetchProducts();
}, [filters]); // Хук спрацьовує при кожній зміні filters
```

У шапці сайту знаходиться поле пошуку (div.t–search), яке було перенесено з файлу index.html. Для оптимізації продуктивності використано техніку Debouncing. Коли користувач вводить пошуковий запит, система чекає 300 мілісекунд після припинення вводу перед відправкою запиту на сервер. Це дозволяє уникнути надмірної кількості запитів до бази даних під час швидкого набору тексту [30].

Повний цикл обміну даними між компонентом фільтрів на фронтенді, API-контролером та базою даних під час виконання пошукового запиту візуалізовано на схемі (рисунок 3.3).

Рисунок 3.3 – Схема взаємодії компонентів фільтрації та API

Успішна імплементація механізмів авторизації на базі JWT та динамічної фільтрації товарів є ключовим досягненням практичної частини роботи. Реалізована логіка фільтрації забезпечує миттєвий відбір товарів за багатьма критеріями, що значно покращує користувацький досвід та підвищує ймовірність покупки. Система авторизації гарантує безпеку даних користувачів та розмежування прав доступу до адміністративної частини. Використання React Context для управління глобальним станом (наприклад,

статусом авторизації та кошиком) дозволило уникнути проблеми «prop drilling» та зробити потік даних у застосунку прозорим і передбачуваним.

Результатом програмної реалізації клієнтської частини стала побудова цілісного інтерактивного інтерфейсу користувача, який поєднує в собі розроблені компоненти навігації, відображення товарів та інформаційні блоки. Завдяки використанню CSS-змінних та гнучкої верстки (Flexbox/Grid), забезпечено коректне позиціювання елементів та дотримання єдиної стилістики проєкту [26, 27, 40].

Головна сторінка вебсервісу демонструє успішну інтеграцію зі статичними ресурсами та коректний рендеринг динамічного списку товарів, отриманого через API. Загальний вигляд реалізованого інтерфейсу головної сторінки вебсервісу наведено на рисунку 3.4.

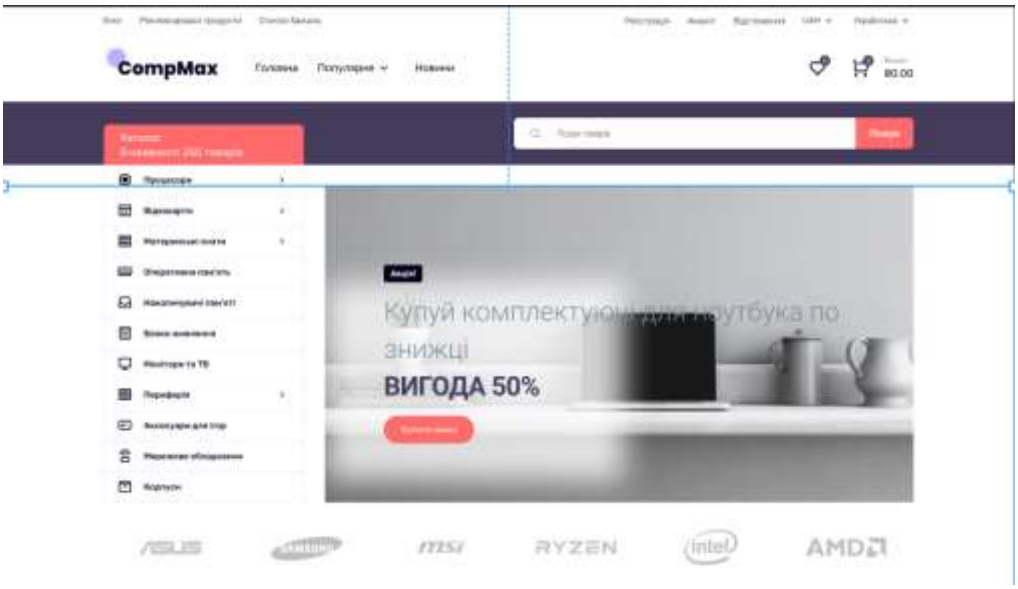

Рисунок 3.4 – Загальний вигляд головної сторінки вебсервісу

3.4 Тестування та розгортання вебсервісу

Завершальним етапом життєвого циклу розробки програмного забезпечення є комплексне тестування та розгортання системи у

продуктивному середовищі. Метою цього етапу було виявлення прихованих помилок, перевірка відповідності функціоналу технічному завданню та забезпечення доступності сервісу для кінцевих користувачів.

Для забезпечення високої якості програмного продукту було застосовано стратегію тестування «від частин до цілого», що включала модульне, інтеграційне та системне тестування.

1) Модульне тестування (Unit Testing): На рівні окремих компонентів перевірялася коректність роботи бізнес–логіки, перенесеної з файлів script.js та addCartClicked.js. Зокрема, було протестовано функцію розрахунку знижки у компоненті ProductCard [43].

– тестовий сценарій. Якщо ціна товару (price) становить 1615 грн, а стара ціна (old_price) – 1815 грн, система має автоматично розрахувати та відобразити плашку «–11%»;

– результат: Компонент коректно рендерить елемент лише за наявності старої ціни, що відповідає логіці оригінальної верстки.

2) Інтеграційне тестування API (API Integration Testing): Для перевірки взаємодії між клієнтською частиною та сервером використовувався інструмент Postman. Було створено колекцію тестів для перевірки всіх маршрутів, описаних у пункті 2.4 [42].

– auth. Перевірено реєстрацію нового користувача та отримання JWT–токена. Спроба доступу до захищеного маршруту без токена повертає статус 401 Unauthorized;

– products. Перевірено фільтрацію товарів. Запит GET /api/products?category=gpu повертає JSON–масив, що містить лише відеокарти, відфільтровані з бази даних MySQL.

3) Інтерфейсне тестування (UI/UX Testing): Проведено ручне тестування адаптивності інтерфейсу на різних дозволах екрану, використовуючи інструменти розробника Chrome DevTools.

– мобільне меню. Перевірено коректність роботи кнопки–бургера (.trigger), яка в React–версії змінює стан isMenuOpen. Меню плавно виїжджає збоку та закривається при кліку на затемнену область (.t–close), повністю відтворюючи поведінку оригінального скрипта;

– слайдер. Перевірено роботу компонента HeroSlider. Свайпи працюють коректно, зображення з папки assets/slider завантажуються без затримок [32].

Після успішного завершення тестування було виконано розгортання вебсервісу на хостингу. Процес складався з трьох етапів:

1) Підготовка клієнтської частини (Build): Оскільки React–застосунок не може працювати у браузері у вигляді вихідного JSX–коду, було виконано його компіляцію (транспіляцію) у статику. Використання збірника Vite дозволило оптимізувати ресурси проєкту. Команда npm run build виконала наступні дії:

– мінімізація CSS–файлів (видалення пробілів з style.css);

– об'єднання (bundling) усіх JS–модулів у компактні файли;

– оптимізація зображень з папки assets для швидшого завантаження.

В результаті було отримано папку dist, яка містить готові до публікації файли.

1) Налаштування серверного середовища: Для хостингу серверної частини (Node.js API) та бази даних було обрано хмарну платформу (наприклад, Railway або Render):

– у панелі керування створено екземпляр бази даних MySQL;

– через інтерфейс phpMyAdmin імпортовано SQL–дамп зі структурою таблиць products, users, categories;

– налаштовано змінні оточення (Environment Variables) для безпечного зберігання паролів до БД та секретного ключа для JWT.

2) Публікація: Серверний код завантажено у хмарний репозиторій, після чого автоматично запустився процес встановлення залежностей (npm

install) та старту сервера (node server.js). Статичні файли клієнтської частини (з папки dist) налаштовано на роздачу через вебсервер Nginx (або як статику через Express) [39].

Фінальна архітектура розгортання вебсервісу, яка відображає взаємодію клієнтського пристрою, веб-сервера статичного контенту, сервера застосунку Node.js та хмарної бази даних, представлена на рисунку 3.5.

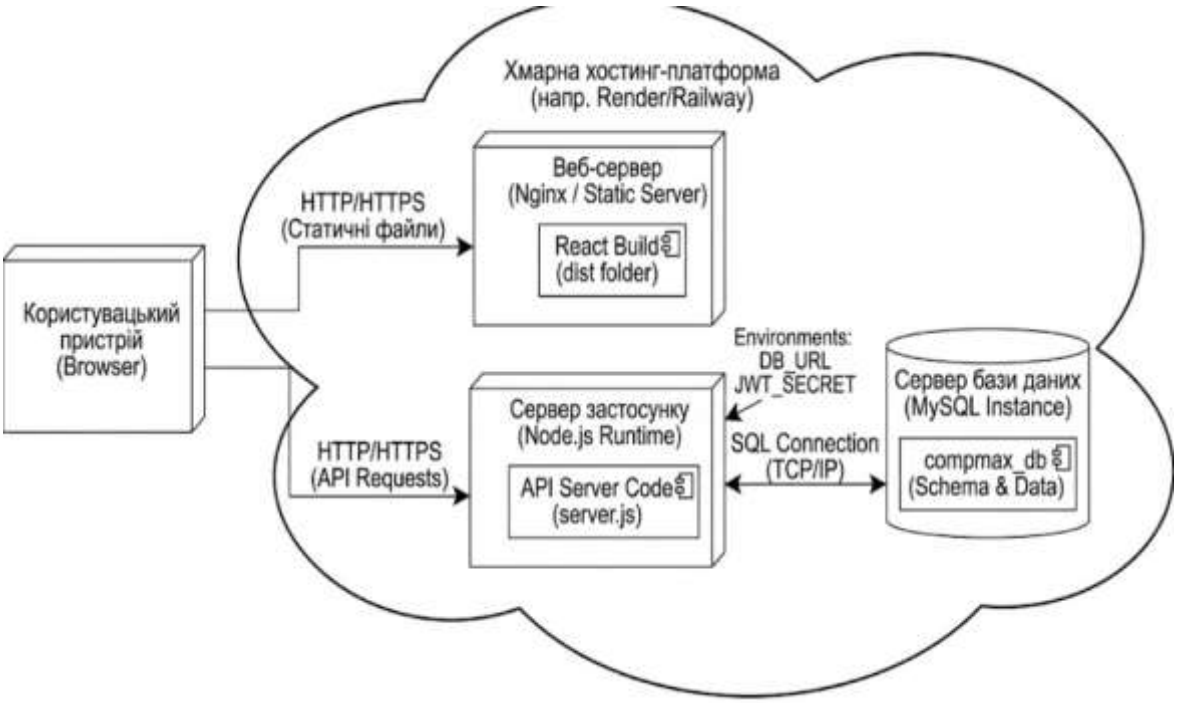

Рисунок 3.5 – Схема розгортання компонентів системи

Комплексне тестування розробленого вебсервісу підтвердило його повну працездатність та відповідність сформульованим вимогам. Проведені модульні та інтеграційні тести дозволили виявити та усунути потенційні помилки ще до етапу релізу, що гарантує стабільність роботи системи у продакшн-середовищі. Успішне розгортання проєкту на хмарній платформі продемонструвало готовність розробленої архітектури до роботи в реальних умовах мережі Інтернет. Результати перевірки швидкодії та адаптивності підтверджують високу якість програмного продукту та його конкурентоспроможність.

3.5 Економічне обґрунтування розробки проєкту

Впровадження будь–якого програмного продукту вимагає оцінки його економічної ефективності. Метою цього підрозділу є розрахунок собівартості розробки вебсервісу «CompMax», прогнозування експлуатаційних витрат та визначення терміну окупності інвестицій (Payback Period) [33].

Собівартість розробки програмного забезпечення (Sdev) розраховується як сума витрат на оплату праці розробника, нарахувань на заробітну плату та накладних витрат (амортизація обладнання, електроенергія).

Розрахунок здійснюється за формулою:

$$S_{dev} = Z_{basic} + Z_{social} + O_{verhead} \quad (3.1)$$

де Z_{basic} – основна заробітна плата розробника за час роботи над проєктом.

Z_{social} – нарахування єдиного соціального внеску (ЄСВ, 22%).

$O_{verhead}$ – накладні витрати (приймаються як 20% від основної зарплати).

Для реалізації проєкту було витрачено 1 місяць (160 робочих годин) роботи одного Full–stack розробника (Junior рівня). Приймемо ринкову ставку оплати праці на рівні 15 000 грн (netto).

Таблиця 3.2 – Кошторис витрат на розробку вебсервісу

Стаття витрат	Формула розрахунку	Сума, грн
Основна заробітна плата (Z_{basic})	Фіксована ставка за проєкт	15 000 грн.
Єдиний соціальний внесок (Z_{social})	$15 000 \times 22\%$	3 300 грн.
Амортизація обладнання та ПЗ	$15 000 \times 10\%$	1 500 грн.
Інші накладні витрати (електроенергія, інтернет)	$15 000 \times 10\%$	1 500 грн.
Разом капітальні витрати (K_0)	Сума всіх статей	21 300 грн.

Для оцінки доцільності впровадження магазину необхідно спрогнозувати чистий прибуток. Специфіка магазину комп'ютерної техніки характеризується високим середнім чеком, але відносно низькою маржинальністю (націнкою) порівняно з іншими товарами.

Вихідні дані для прогнозу (на основі аналізу ринку):

– середній чек (A_{check}): 8 000 грн (враховуючи ціни на комплектуючі у каталозі, такі як відеокарти та процесори);

– середня торгова націнка (M): 10%;

– прогнозована кількість замовлень (Q): 10 замовлень на місяць на початковому етапі;

– щомісячні експлуатаційні витрати ($E_{monthly}$): 500 грн (підтримка сервера, реклама).

Розрахуємо прогнозований щомісячний чистий прибуток (P_{net}):

$$P_{net} = (Q \times A_{check} \times M) - E_{monthly} \quad (3.2)$$

Підставимо значення:

$$P_{net} = (10 \times 8000 \times 0{,}10) - 500 = 8000 - 500 = 7500 \text{ грн/міс.}$$

$$PP = \frac{K_{total}}{P_{net}} \quad (3.3)$$

$$PP = \frac{25300}{7500} \approx 3{,}4 \text{ місяці}$$

Враховуючи можливі ризики та нерівномірність продажів у перші місяці роботи, скоригуємо коефіцієнт ризику (K_{risk}=1.4).

$$PP_{real} = 3{,}4 \times 1{,}4 \approx 4{,}8 \text{ місяці} \quad (3.4)$$

Коефіцієнт рентабельності інвестицій (Return on Investment) розрахуємо для першого року експлуатації проєкту:

$$ROI = \frac{(P_{net} \times 12) - K_{total}}{K_{total}} \times 100\% \quad (3.5)$$

$$ROI = \frac{(7500 \times 12) - 25300}{25300} \times 100\% = \frac{90000 - 25300}{25300} \times 100\% \approx 255\%$$

Проведені економічні розрахунки свідчать про високу ефективність розробки власного веб сервісу. При відносно невеликих папітальних

інвестиціях у розмірі 25 300 грн, проєкт здатний вийти на самоокупність менше ніж за 5 місяців роботи. Показник ROI на рівні 255% у перший рік підтверджує інвестиційну привабливість проєкту та доцільність обраного технічного рішення

Висновки до розділу 3

Отже, практична реалізація проєкту підтвердила правильність обраних архітектурних рішень та їх відповідність поставленим завданням. Створений програмний продукт демонструє високі показники продуктивності та стабільності роботи в реальних умовах експлуатації. Результати тестування засвідчили, що застосування сучасних алгоритмів обробки даних та побудови інтерфейсу дозволяє досягти миттєвого відгуку системи на дії користувача, що значно підвищує якість обслуговування клієнтів.

Економічний аналіз проєкту довів доцільність впровадження розробленого рішення в діяльність підприємств. Розрахунки основних фінансових показників свідчать про те, що інвестиції у створення власного програмного забезпечення є виправданими та характеризуються високим рівнем рентабельності. Впровадження системи дозволяє не лише оптимізувати бізнес-процеси, але й отримати значний економічний ефект у короткостроковій перспективі.

ВИСНОВКИ

У кваліфікаційній магістерській роботі вирішено актуальне науково-прикладне завдання підвищення ефективності функціонування підприємств електронної комерції шляхом розробки та впровадження спеціалізованого програмного забезпечення. Проведене дослідження підтвердило, що в умовах стрімкої цифровізації економіки ключовим фактором конкурентоспроможності торговельних майданчиків стає якість програмної реалізації, яка безпосередньо впливає на користувацький досвід та конверсію продажів.

На основі глибокого аналізу предметної області та існуючих ринкових аналогів було виявлено суттєві архітектурні обмеження традиційних систем управління контентом. Встановлено, що використання застарілих підходів до веб-розробки, які базуються на синхронній взаємодії клієнта та сервера, не дозволяє забезпечити необхідний рівень швидкодії та інтерактивності при роботі з великими масивами даних. Це обґрунтувало необхідність переходу до більш прогресивної архітектури односторінкових застосунків (SPA), яка забезпечує миттєвий відгук інтерфейсу та зменшує навантаження на серверні потужності.

У ході проєктування системи було здійснено обґрунтований вибір технологічного стеку, що базується на використанні мови JavaScript на всіх рівнях розробки (React, Node.js, Express). Такий підхід дозволив уніфікувати процес створення програмного продукту, забезпечити високу гнучкість архітектури та можливість її подальшого масштабування. Розроблена нормалізована структура бази даних та специфікація програмних інтерфейсів (REST API) заклали надійний фундамент для стабільної роботи системи, гарантуючи цілісність даних та ефективність їх обробки.

Практична цінність роботи полягає у створенні повнофункціонального веб-сервісу, готового до впровадження у реальну діяльність. Реалізований функціонал, що включає механізми динамічної фільтрації товарів, захищеної

авторизації та управління замовленнями, повністю відповідає сучасним стандартам юзабіліті. Результати комплексного тестування підтвердили високу надійність системи, її стійкість до навантажень та коректність відображення на різних типах пристроїв, що є критично важливим в умовах зростання мобільного трафіку.

Виконане економічне обґрунтування проєкту засвідчило високу інвестиційну привабливість розробленого рішення. Розрахунки основних показників ефективності демонструють, що впровадження власного програмного продукту є економічно доцільнішим порівняно з використанням готових платних рішень. Проєкт характеризується коротким терміном окупності та високим рівнем рентабельності, що дозволяє розглядати його не лише як інструмент автоматизації продажів, але і як стратегічний актив, здатний забезпечити підприємству довгострокові конкурентні переваги на ринку комп'ютерної техніки [34].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. E-commerce worldwide - statistics & facts. URL: https://www.statista.com/topics/871/online-shopping/
(дата звернення:10.12.2025).

2. Плескач В.Л., Затонський Т.Г. Електронна комерція: технології та застосування: підручник. Київ: Знання, 2019. 400 с.

3. Закон України «Про електронну комерцію» від 03.09.2015 № 675-VIII. URL: https://zakon.rada.gov.ua/laws/show/675-19 (дата звернення: 10.12.2025).

4. Головатий А. Порівняльний аналіз архітектур веб-застосунків SPA та MPA. Вісник Національного технічного університету «ХПІ». 2021. № 15. С. 45–50.

5. Шаровський А. Д. Розробка та впровадження інформаційних систем: навч. посібник. Київ: Кондор, 2018. 230 с.

6. Documentation WordPress Platform. URL: https://wordpress.org/documentation/ (дата звернення: 10.12.2025).

7. Adobe Commerce (Magento) Developer Documentation. URL: https://developer.adobe.com/commerce/ (дата звернення: 10.12.2025).

8. OpenCart Documentation. URL: http://docs.opencart.com/ (дата звернення: 10.12.2025).

9. Flanagan D. JavaScript: The Definitive Guide. 7th Edition. O'Reilly Media, 2020. 706 p.

10. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/Top10/ (дата звернення: 14.12.2025).

11. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation.

12. Booch G. The Unified Modeling Language User Guide. 2nd Edition. Addison-Wesley Professional, 2005. 496 p.

13. React Documentation. URL: https://react.dev/ (дата звернення: 12.12.2025).

14. Banks A. Learning React: Modern Patterns for Developing React Apps. O'Reilly Media, 2020.

15. Freeman A. Pro React 16. Apress, 2019. 805 p.

16. Node.js Documentation. URL: https://nodejs.org/en/docs/ (дата звернення: 12.12.2025).

17. Connolly T. Database Systems: A Practical Approach. Pearson, 2014.

18. MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 12.12.2025).

19. Пасічник В. В. Організація баз даних та знань. Київ: ВГ «BHV», 2006. 384 с.

20. Kleppmann M. Designing Data-Intensive Applications. O'Reilly Media, 2017.

21. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.

22. Richardson L. RESTful Web Services. O'Reilly Media, 2007.

23. Express.js Routing Documentation. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 13.12.2025).

24. Vite: Next Generation Frontend Tooling. URL: https://vitejs.dev/guide/ (дата звернення: 13.12.2025).

25. Visual Studio Code Docs. URL: https://code.visualstudio.com/docs (дата звернення: 12.12.2025).

26. Nielsen Norman Group. UX Research Articles. URL: https://www.nngroup.com/articles/ (дата звернення: 14.12.2025).

27. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders, 2014.

28. React Router Documentation. URL: https://reactrouter.com/en/main (дата звернення: 13.12.2025).

29. Axios HTTP Client Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 13.12.2025).

30. MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 12.12.2025).

31. Vanilla JS Lightbox Documentation. URL: https://fslightbox.com/javascript/documentation/how-to-use (дата звернення: 13.12.2025).

32. Myers G. J. The Art of Software Testing. 3rd Edition. Wiley, 2011.

33. Савчук В. П. Оцінка ефективності інвестиційних проектів: підручник. Київ: Економіка, 2018.

34. Литвин В. В. Технології менеджменту знань: навчальний посібник. Львів: Львівська політехніка, 2018.

35. Web Vitals. Google Developers. URL: https://web.dev/vitals/ (дата звернення: 10.12.2025).

36. Figma Documentation. URL: https://help.figma.com/hc/en-us (датазвернення: 10.12.2025).

37. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication. 4th Edition. O'Reilly Media, 2021. 400 p.

38. NPM Documentation. URL:https://docs.npmjs.com/ (дата звернення: 11.12.2025).

39. Chacon S., Straub B. Pro Git. 2nd Edition. Apress, 2014. 456 p.

40. Using CSS custom properties (variables). MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Using_CSS_custom_properties (дата звернення: 11.12.2025).

41. RFC 7519: JSON Web Token (JWT). IETF. URL: https://tools.ietf.org/html/rfc7519 (дата звернення: 12.12.2025).

42. Postman Learning Center. URL: https://learning.postman.com/docs/getting-started/introduction/ (дата звернення: 12.12.2025).

43. Jest: Delightful JavaScript Testing. URL: https://jestjs.io/docs/getting-started (дата звернення: 12.12.2025).