Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ВЕБМАРКЕТПЛЕЙСУ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ З МОЖЛИВІСТЮ СТВОРЕННЯ КОРИСТУВАЦЬКИХ МАГАЗИНІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-1
Глумаков В. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С. В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Глумакову Владиславу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебмаркетплейсу для електронної комерції з можливістю створення користувацьких магазинів

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 7 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, мова програмування TypeScript, бібліотека React, фреймворк Next.js, платформа Node.js, фреймворк Fastify, метафреймворк NestJS, програмне середовище розробки JetBrains WebStorm.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних маркетлпейсів.
2. Дослідити особливості та засоби реалізації.
3. Моделювання структури вебмаркетплейсу.
4. Реалізація сучасного маркетплейсу для електронної комерції.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми вебмаркетплейсів, постановка задачі, зображення дизайну, блок-схеми архітектури.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
13	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Машталір С. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 67 с., 5 табл., 22 рис., 30 джерел.

ЕЛЕКТРОННА КОМЕРЦІЯ, ВЕБМАРКЕТПЛЕЙС, БАЗА ДАНИХ, КОРИСТУВАЦЬКИЙ МАГАЗИН, ІНТЕРФЕЙС КОРИСТУВАЧА, СЕРВЕРНА ЧАСТИНА, КЛІЄНТСЬКА ЧАСТИНА.

Об’єктом роботи є вебсистема маркетплейсу для електронної комерції.

Метою роботи є розробка вебзастосунку маркетплейсу, який дозволяє користувачам створювати власні магазини, керувати товарами та здійснювати онлайн-продажі. У рамках роботи спроєктовано архітектуру клієнтської та серверної частин застосунку, створено зручну та ефективну базу даних для зберігання інформації про користувачів, товари, замовлення та доставки.

У процесі роботи проаналізовано існуючі рішення у сфері електронної комерції, обґрунтовано вибір технологій для реалізації функціоналу маркетплейсу. Розроблено інтерфейс користувача, реалізовано можливість фільтрації товарів за категоріями, додавання та редагування товарів, обробки замовлень.

У результаті роботи створено повнофункціональний вебзастосунок з інтуїтивно зрозумілим інтерфейсом та стабільною серверною логікою, що забезпечує ефективну взаємодію між продавцями та покупцями.

E-COMMERCE, WEB MARKETPLACE, DATABASE, USER STORE, USER INTERFACE, BACKEND, FRONTEND.

The object of the work is a web-based marketplace system for e-commerce.

The aim of the work is to develop a web application that allows users to create their own online stores, manage products, and conduct sales via the Internet. The project includes the design of both client-side and server-side architecture, along with the development of an efficient database to store information about users, products, orders, and deliveries.

The research involves analysis of existing e-commerce platforms, justification for selected technologies, and the implementation of key marketplace functionalities. A user-friendly interface was developed, supporting product filtering, item creation and editing, and order processing.

As a result, a fully functional web application was created, offering an intuitive interface and robust backend logic, enabling seamless interaction between sellers and customers.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІС – інформаційна система

CRM – Customer Relationship Management (управління взаємовідносинами з клієнтами)

CMS – Content Management System (система керування вмістом)

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

SPA – Single Page Application (односторінковий застосунок)

API – Application Programming Interface (прикладний програмний інтерфейс)

REST – Representational State Transfer (передача репрезентативного стану)

СКБД – система керування базами даних

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту)

SOLID – Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle (принцип єдиної відповідальності, принцип відкритості/закритості, принцип заміщення Лісков, принцип розділення інтерфейсу, принцип інверсії залежностей)

ВСТУП

У сучасному суспільстві Інтернет відіграє надзвичайно важливу роль, забезпечуючи доступ до інформації в будь-якому місці та в будь-який час. Його активне використання сприяє розвитку численних сфер діяльності, зокрема електронної комерції, яка стала невід'ємною частиною цифрової економіки.

Маркетплейси як різновид платформ для електронної торгівлі дають змогу продавцям реалізовувати товари та послуги онлайн, а покупцям — здійснювати зручні й швидкі покупки незалежно від географічного розташування. Такий формат значно розширює потенціал ринку й відкриває нові можливості для підприємців і користувачів.

Ключовим елементом функціонування подібних ІС є база даних, що відповідає за структуроване збереження інформації про товари, категорії, замовлення, користувачів та адреси доставки. Ефективне суправління цією інформацією є запорукою стабільної та швидкої роботи всього вебзастосунку.

Метою цієї роботи є аналіз предметної області та розробка серверної і клієнтської частин вебмаркетплейсу, який дає можливість користувачам створювати власні магазини. Проєкт передбачає створення зручної бази даних для зберігання основної інформації, реалізацію зручного інтерфейсу та забезпечення безпеки користувацьких даних.

Результатом розробки стане функціональний вебзастосунок, що забезпечить простоту у використанні, швидкий пошук товарів, розширення асортименту, а також сприятиме зростанню кількості користувачів та посиленню позицій платформи на ринку електронної комерції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз вимог до платформи електронної комерції

Фундаментальним етапом для створення інформаційної системи електронної комерції є глибокий аналіз предметної області, чітке структурування робочих процесів та розуміння послідовності етапів розробки [1]. Не менш суттєвим є вивчення потреб цільової аудиторії, адже це дозволяє побудувати не просто робочу, а дійсно ефективну та інтуїтивно зрозумілу платформу для онлайн-продажів.

Для успішної реалізації проекту, компанії та їхні команди, відповідальні за створення компонентів системи електронної комерції, мають зосередитись на таких основних завданнях:

- аналіз ринку та конкурентного середовища: необхідно провести аналіз ринкових тенденцій, цільових клієнтів, конкурентів та їхніх стратегій. Це дозволить врахувати ринкові вимоги та знайти конкурентні переваги при розробці платформи;
- формування вимог до платформи: важливо чітко сформулювати функціональні потреби ІС: від інструментів керування бізнесом та обробки замовлень до функцій кошика, оплати та можливостей інтеграції з іншими необхідними сервісами;
- розробка зручного інтерфейсу: створення інтуїтивно зрозумілого та зручного інтерфейсу є критично важливим. Він має забезпечувати легкий пошук товарів та простий процес замовлення, що безпосередньо впливає на користувацький досвід та конверсію;
- створення системи управління контентом: потрібна гнучка система, що дозволяє легко та ефективно керувати каталогом товарів, оновлювати інформацію та підтримувати комунікацію з клієнтами через різні канали (соціальні мережі, email тощо);

– розробка модулів обробки замовлень і платежів: необхідно створити надійну та безпечну систему, яка дозволяє клієнтам легко розміщувати замовлення, відслідковувати їх статус та зручно здійснювати оплату різними поширеними методами;

– впровадження системи аналітики та звітності: потрібна система для систематичного збору та аналізу даних про продажі, трафік та поведінку користувачів. Це допомагає приймати обґрунтовані бізнес-рішення та підвищувати загальну ефективність [2];

– забезпечення кібербезпеки та захисту даних: впровадження надійних заходів безпеки для захисту персональних та фінансових даних клієнтів є обов'язковим для здобуття їхньої довіри та відповідності регуляторним вимогам [3];

– інтеграція з внутрішніми та зовнішніми системами: ефективне поєднання з ключовими зовнішніми та внутрішніми системами (наприклад, управління складом, логістика, CRM) підвищує загальну ефективність операційних процесів та зручність управління маркетплейсом;

– маркетинговий функціонал платформи: платформа має надавати можливості для створення та проведення цільових маркетингових кампаній (включно з промоакціями, знижками, програмами лояльності) для залучення визначеної аудиторії та стимулювання продажів.

У контексті розробки власної CMS-системи з використанням монолітної архітектури особливої актуальності набуває питання інтеграції всіх функціональних блоків у єдину програмну структуру. Такий підхід дозволяє мінімізувати накладні витрати, зменшити складність деплойменту та спростити підтримку системи на ранніх етапах. Проте, важливо забезпечити чітке розмежування логіки за допомогою внутрішніх модулів, що дозволяє підтримувати масштабованість і розширюваність без переходу на мікросервісну архітектуру.

Не менш важливою складовою є користувацький інтерфейс. UX-дизайн повинен відповідати очікуванням цільової аудиторії, забезпечуючи просту

навігацію, логічну структуру вмісту та мінімізацію дій для досягнення цілей користувача. Зручність використання платформи напряму впливає на кількість замовлень та повернення клієнтів, що робить UX-орієнтоване проєктування критично важливим етапом при створенні маркетплейсів.

Підсумовуючи, створення маркетплейсу в рамках електронної комерції є комплексним процесом, що об'єднує різноманітні бізнес-функції та процеси для забезпечення повноцінної роботи всієї інформаційної системи [4]. Основними цілями при розробці такого майданчика незмінно залишаються створення зручного та привабливого інтерфейсу, забезпечення широкого та актуального асортименту товарів, а також гарантування максимальної безпеки даних користувачів. Досягнення цих цілей вимагає залучення команди кваліфікованих спеціалістів, які володіють сучасними технологіями, та постійного відстеження інновацій і своєчасного оновлення для підтримки актуальності та конкурентоспроможності платформи. Окрему увагу слід приділити розробці та підтримці мобільних застосунків, зважаючи на стійке зростання кількості користувачів, які здійснюють покупки зі смартфонів та планшетів, щоб забезпечити їм зручність замовлення з будь-яких пристроїв [5]. Таким чином, успішна розробка системи електронної комерції потребує системного, продуманого підходу та реалізації широкого спектру функцій задля забезпечення швидкої, дієвої та зручної роботи сучасного онлайн-магазину чи маркетплейсу [6].

1.2 Аналіз існуючих маркетплейс систем

У сучасному цифровому просторі існує велика кількість онлайн-маркетплейсів, які забезпечують зручний механізм для здійснення купівлі та продажу товарів різноманітних категорій. Такі платформи поєднують в собі функції інтернет-магазину та торгового майданчика, дозволяючи продавцям розміщувати свою продукцію, а покупцям – знаходити й замовляти необхідні

товари в межах одного вебзастосунку. З огляду на актуальність теми електронної комерції, в рамках цієї роботи особлива увага приділяється аналізу існуючих рішень, що вже реалізовані та активно використовуються на українському ринку.

Для більш ґрунтовного порівняльного аналізу та формування вимог до власного вебмаркетплейсу було обрано два популярні сервіси – Prom та Rozetka. Обидві платформи мають широкий функціонал, високий рівень популярності серед користувачів та схожі принципи організації структури торгівлі онлайн. Аналіз їхніх переваг, недоліків і підходів до реалізації основних процесів дозволить сформувати уявлення про сильні та слабкі сторони сучасних маркетплейс-систем, а також використати отримані висновки як орієнтир при проєктуванні та реалізації власного рішення.

1.2.1 Аналіз маркетплейсу Prom

Prom є одним із найбільших українських онлайн-маркетплейсів, що надає платформу для продажу товарів та послуг як малим, так і середнім підприємствам. Сервіс орієнтований на створення сприятливих умов для електронної комерції, дозволяючи продавцям відкривати власні онлайн-магазини, а покупцям – зручно здійснювати замовлення через єдиний інтерфейс.

Платформа орієнтована на інтуїтивну взаємодію з користувачем та надає великий вибір категорій товарів, доступних для пошуку та покупки. Однією з важливих особливостей Prom є можливість не тільки купувати товари, а й порівнювати їх, читати відгуки покупців, переглядати рейтинги продавців, що значно спрощує процес вибору.

Для початку роботи з маркетплейсом користувачу необхідно відвідати офіційний вебсайт платформи та ознайомитися з її ключовими функціями. На

рисунку 1.1 представлено головну сторінку інтерфейсу Prom, яка демонструє основні елементи навігації та доступні можливості для здійснення покупок.

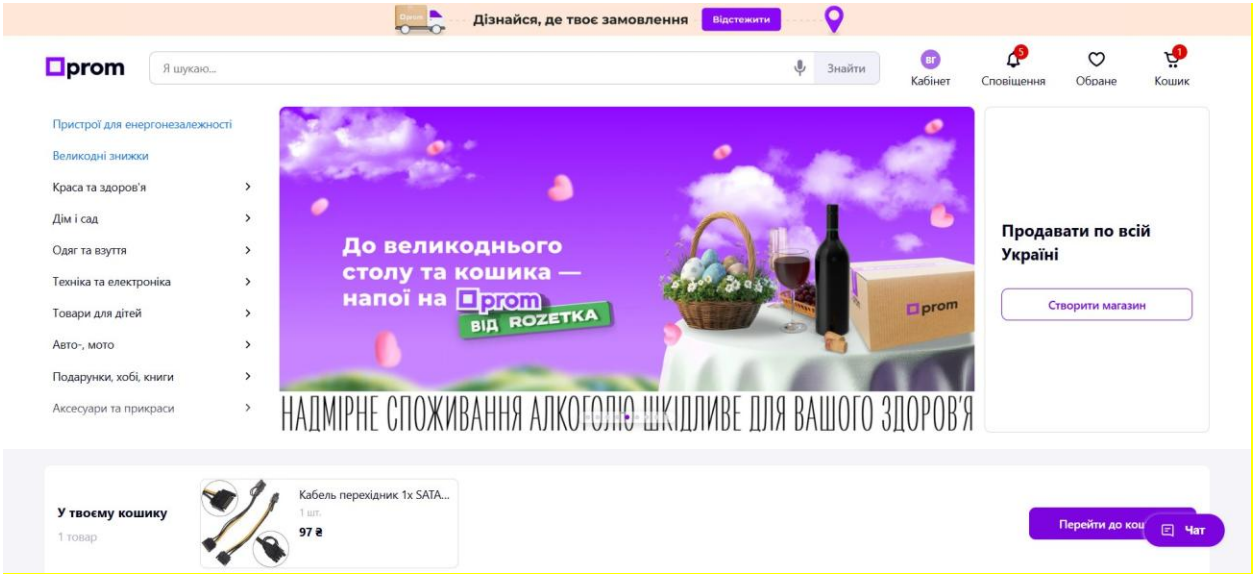

Рисунок 1.1 – Головна сторінка сайту Prom

На головній сторінці сайту ми бачимо такі елементи:

- великий банер з поточними акціями;
- кошик покупця з індикацією кількості товарів, які вже додано;
- кнопка сповіщень, де можна дізнатися про акційні пропозиції та зміни статусів доставки;
- кнопка обране, щоб перейти до списку збережених товарів;
- кнопка особистого кабінету, яка і одночасно є кнопкою реєстрації та входу;
- рядок пошуку, що дозволяє знаходити товари за їхньою назвою;
- список з категоріями та підкатегоріями;
- перелік рекомендованих та переглянутих товарів.

Для початку повноцінного користування всіма можливостями платформи, необхідно зареєструватися. Prom підтримує кілька варіантів авторизації: через інтегрований акаунт Google або класичну реєстрацію за допомогою електронної пошти чи мобільного номера. Після введення даних потрібно підтвердити особу шляхом введення коду, надісланого у

відповідному форматі. Увійшовши до системи користувач має ряд додаткових функцій для перегляду, який наведений на рисунку 1.2

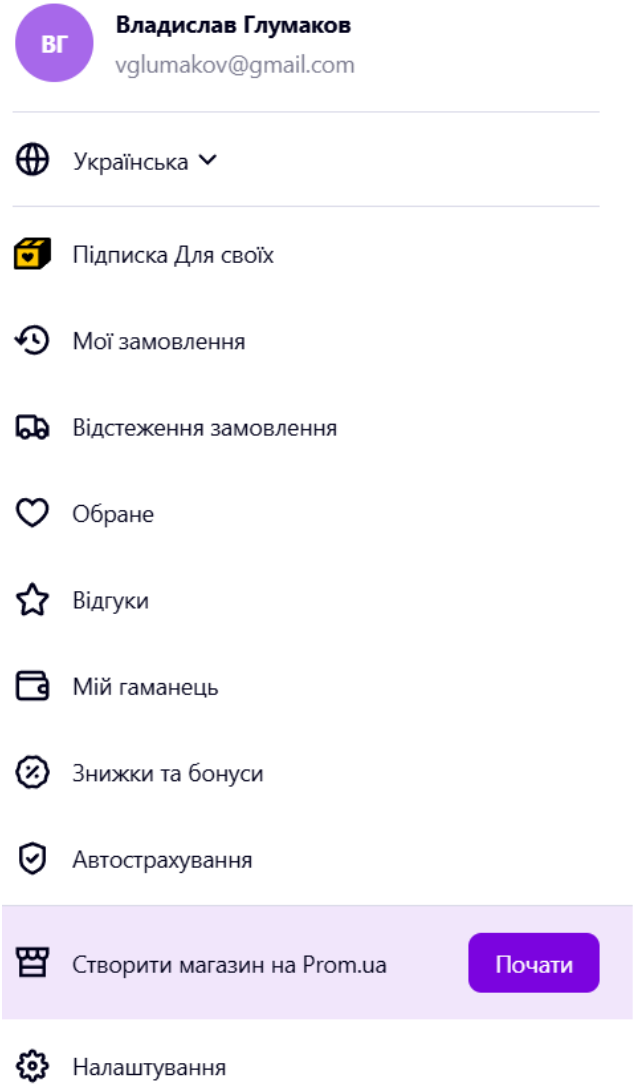

Рисунок 1.2 – Функції авторизованого користувача

Щоб додати товар до кошика, спершу потрібно перейти до відповідної категорії. Наприклад, обравши категорію «Монітори», користувач бачить широкий вибір моделей від різних виробників із зазначенням основних характеристик, цін та наявності на складі. Після вибору товару можна ознайомитися з його описом, рейтингами та відгуками, а також натиснути кнопку «Купити», щоб додати його до кошика. Перелік доступних товарів та щойно доданий товар до кошика зображено на рисунку 1.3.

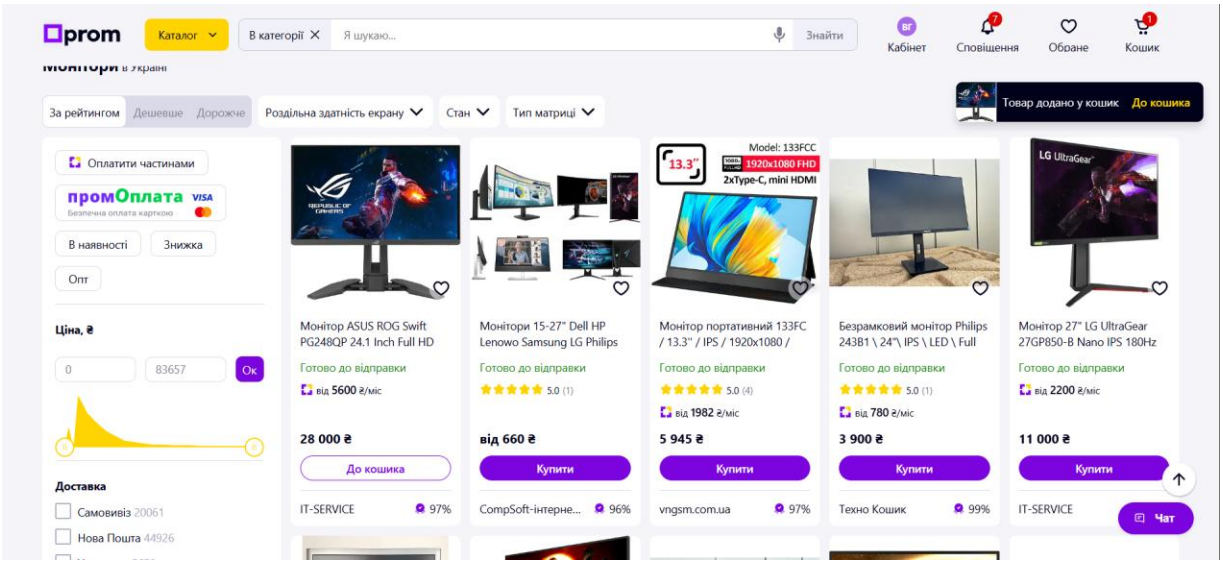

Рисунок 1.3 – Доданий товар та каталог товарів за категорією «Монітори»

Коли всі необхідні товари додано до кошика, слід перейти на сторінку замовлення. Для цього потрібно натиснути кнопку «Оформити замовлення». Після цього система перенаправляє користувача на сторінку з формами для заповнення персональних і контактних даних, зображену на рисунку 1.4.

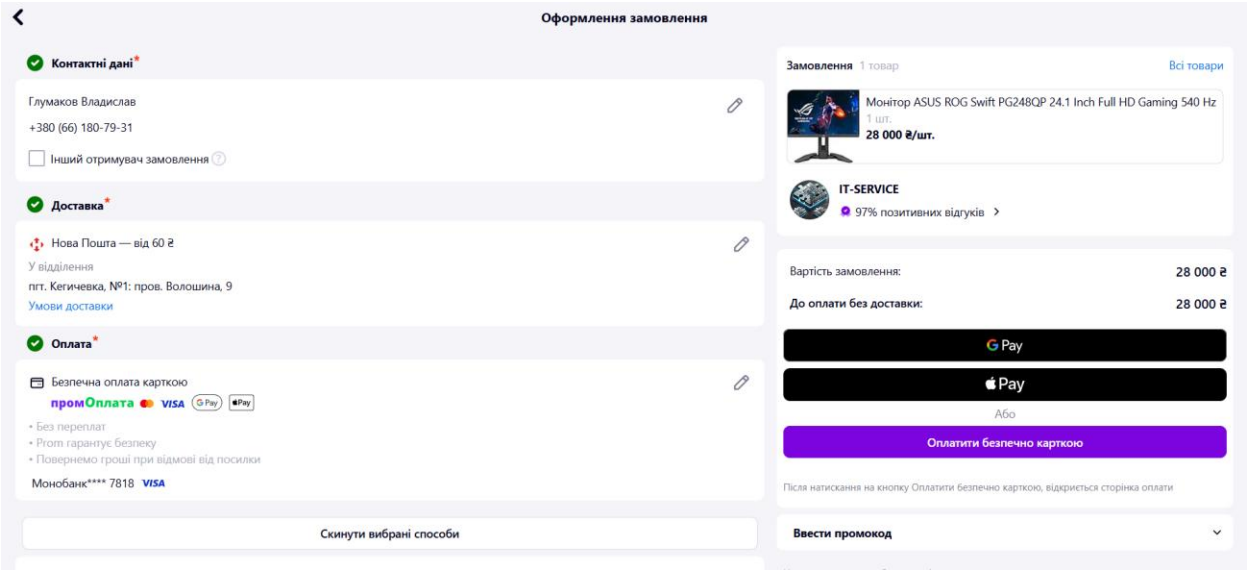

Рисунок 1.4 – Сторінка оформлення замовлення

На цій сторінці представлено кілька форм, які користувач повинен послідовно заповнити для успішного оформлення замовлення. Кожна з них

містить поля для внесення важливої інформації, необхідної для ідентифікації покупця, організації доставки та проведення оплати. Ці форми умовно поділяються на три основні блоки:

- контактні дані;
- доставка;
- оплата.

У першій формі користувачу потрібно ввести своє прізвище, ім’я та номер телефону. Це потрібно для зв’язку з покупцем та підтвердження особи. Після введення номера система надсилає код підтвердження, який слід ввести у відповідне поле для проходження верифікації. Далі користувач переходить до розділу доставки, де потрібно вказати населений пункт та обрати зручний спосіб доставки – наприклад, через службу Нова пошта. Також необхідно зазначити номер відділення або, у разі адресної доставки, – точну адресу. У деяких випадках можна додати коментарі щодо часу чи способу вручення товару.

Далі слід перейти до розділу оплати, де необхідно вибрати зручний спосіб розрахунку та заповнити відповідні платіжні дані. Також користувач має можливість скористатися швидкими платіжними сервісами, такими як Google Pay або Apple Pay.

Таким чином, ознайомившись із основними етапами користування платформою Prom та її функціональними можливостями, можна виділити ключові переваги та недоліки цього сервісу:

Переваги:

- доступність для малого бізнесу: низький поріг входу – для реєстрації достатньо номера телефону та електронної пошти [7];
- можливість створення власного магазину: продавці можуть створити персоналізований сайт з унікальним дизайном, банерами та брендуванням;
- велика аудиторія: понад 44 млн відвідувань на місяць, що перевищує показники Rozetka [8];

– гнучкі маркетингові інструменти: можливість налаштування внутрішньої реклами та SEO-оптимізації.

Недоліки:

– висока конкуренція: багато продавців, що призводить до цінової конкуренції та демпінгу;

– обмежена кастомізація: шаблонні сайти з обмеженими можливостями зміни дизайну;

– додаткові витрати: платна реклама та просування товарів можуть значно збільшити витрати продавця [9].

1.2.2 Аналіз маркетплейсу Rozetka

Rozetka – одна з найпопулярніших платформ електронної комерції в Україні, яка поєднує функціональність маркетплейсу та класичного інтернет-магазину. Платформа орієнтована як на кінцевого споживача, так і на бізнес, який прагне продавати свою продукцію через перевірений і масштабний онлайн-канал.

Щоб розпочати роботу з маркетплейсом Rozetka, необхідно перейти до його вебзастосунку та ознайомитися з основними функціональними можливостями. На рисунку 1.5 зображено головну сторінку інтерфейсу.

На стартовій сторінці платформи можна побачити такі елементи:

– кошик покупця з індикацією кількості товарів, які вже додано;

– кнопка сповіщень, де можна дізнатися про акційні пропозиції та зміни статусів доставки;

– кнопка для входу або реєстрації нового користувача;

– рядок пошуку, що дозволяє знаходити товари за їхньою назвою;

– кнопка переходу до каталогу, який містить структуру з категоріями та підкатегоріями;

– перелік основних товарних розділів;

– банери з поточними акційними пропозиціями та знижками.

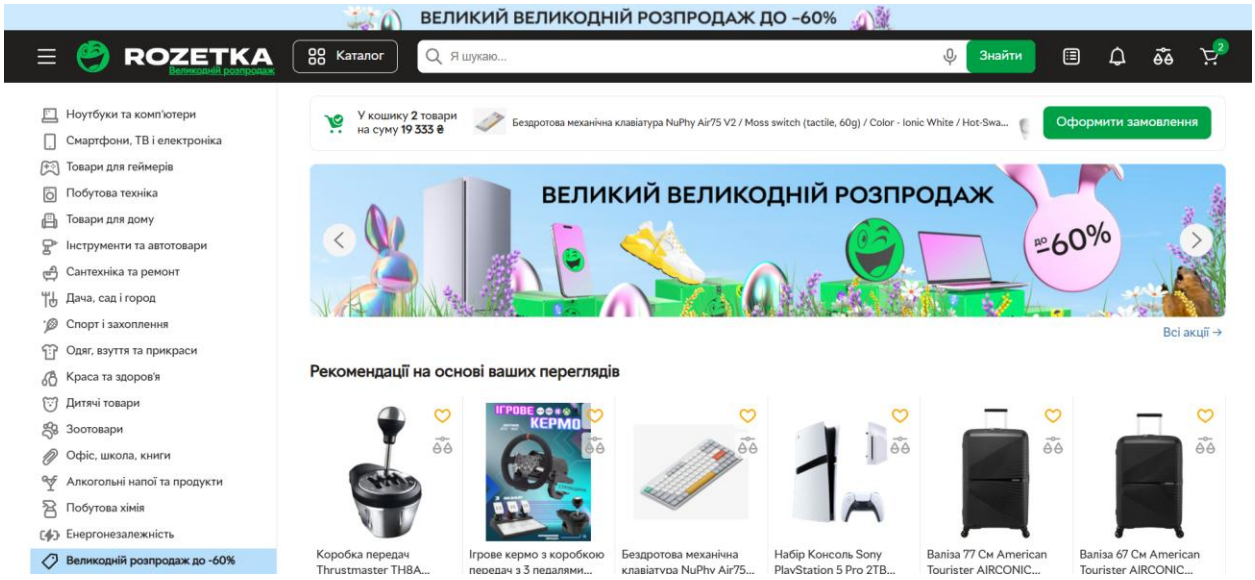

Рисунок 1.5 – Головна сторінка сайту Rozetka

Початковим кроком під час здійснення покупки є вибір потрібного товару та його додавання до кошика. Для продовження оформлення покупки слід перейти на сторінку кошика та натиснути кнопку «Оформити замовлення». Після цього система перенаправить користувача на сторінку оформлення замовлення, приклад якої представлено на рисунку 1.6

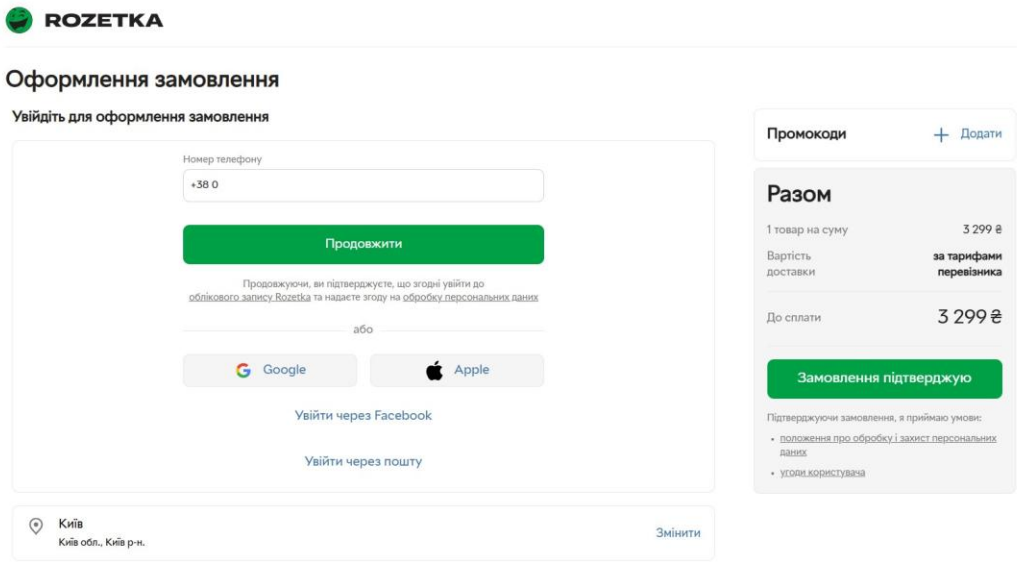

Рисунок 1.6 – Сторінка оформлення замовлення

На цій сторінці можна увійти до облікового запису за допомогою номеру телефону, електронної пошти або через акаунти Google, Apple чи Facebook. Після авторизації система автоматично підтягне збережену інформацію користувача, таку як контактний номер, адреса доставки та e-mail. Водночас, якщо користувач не має зареєстрованого акаунта або не бажає проходити реєстрацію, він може самостійно внести необхідні дані та завершити оформлення замовлення як гість.

Після входу до системи користувач отримує доступ до свого особистого кабінету, де зібрана додаткова інформація (рис. 1.7), зокрема:

- мій акаунт Rozetka;
- особисті дані;
- мої отримувачі замовлень;
- контакти;
- адреса доставки;
- захоплення;
- домашні тварини.

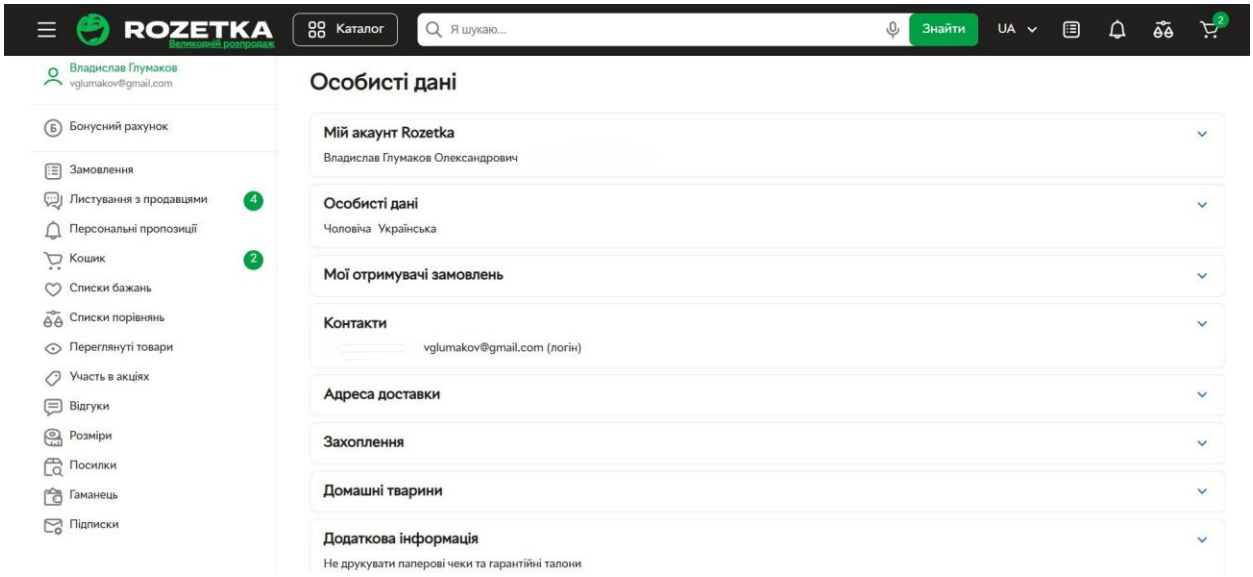

Рисунок 1.7 – Додаткова інформація особистого профілю користувача в системі після авторизації

Після детального розгляду особливостей роботи маркетплейсу Rozetka, стає можливим визначити основні переваги та недоліки цієї платформи, які варто враховувати при створенні власного вебрішення:

Переваги:

- висока довіра споживачів: Rozetka є одним з найвідоміших брендів в Україні, що підвищує довіру до продавців на платформі;
- активне просування товарів: платформа самостійно рекламує товари продавців, використовуючи ремаркетинг та інші інструменти;
- менша конкуренція: порівняно з Prom, на Rozetka менше продавців, що може бути вигідно для новачків.

Недоліки:

- висока комісія: комісія за продажі може становити від 5% до 20%, що знижує прибутковість;
- складна реєстрація: процес реєстрації може займати від 2 тижнів до 2 місяців і вимагає наявності ФОП або ТОВ;
- обмеження в брендуванні: немає можливості просувати власний бренд, використовувати логотип та назву магазину [10];
- обмежені можливості реклами: не можна підключити сторонню рекламу; платформа самостійно просуває товари.

1.3 Функціональне призначення та галузі застосування системи

Розроблювана ІС електронної комерції типу маркетплейс призначена для створення зручного онлайн-середовища, яке забезпечує ефективну взаємодію між продавцями та покупцями. Така система може знайти застосування у різних сферах бізнесу – від роздрібної та оптової торгівлі до виробництва, дистрибуції та навіть сфери послуг. Вона дозволяє підприємствам різного масштабу цифровізувати свою діяльність, забезпечуючи централізоване управління усіма ключовими процесами.

За допомогою маркетплейсу можна автоматизувати важливі бізнес-операції, зокрема, керування товарним асортиментом, обробку замовлень, прийом оплат, організацію доставки, а також підтримку актуального стану інформації щодо товарів, цін та наявності. Крім того, система забезпечує збереження та обробку персональних даних користувачів, їх ідентифікацію, облік активності, а також дає можливість створювати звіти для аналітики продажів і клієнтської взаємодії.

Серед основних можливостей платформи – підтримка створення індивідуальних інтернет-магазинів для кожного продавця. Це дозволяє брендам формувати власну вітрину всередині маркетплейсу, розвивати унікальний імідж і розширювати клієнтську базу. Завдяки можливості інтеграції з соціальними мережами, сторонніми сервісами доставки та платіжними платформами, система стає ще зручнішою для кінцевих користувачів.

Ключовою бізнес-функцією є організація продажу товарів в онлайн-режимі. Це відкриває перед підприємцями можливість виходу за межі локального ринку та залучення клієнтів з будь-якої географічної точки. У підсумку, використання такої системи сприяє підвищенню рівня обслуговування, збільшенню обсягів продажу та розширенню ринку збуту, що робить її актуальним інструментом для сучасного бізнесу

1.4 Постановка задачі

Таким чином, створення зручного та функціонального вебмаркетплейсу є актуальним завданням у сфері електронної комерції, оскільки дає змогу користувачам ефективно здійснювати купівлю та продаж товарів у мережі. Тому ставиться завдання розробки вебзастосунку маркетплейсу з реалізацією функціоналу індивідуальних магазинів для

продавців, інтегрованою системою управління товарами, категоріями та замовленнями.

Об’єктом роботи є вебсистема маркетплейсу для електронної комерції.

Метою роботи є розробка вебзастосунку маркетплейсу, який дозволяє користувачам створювати власні магазини, керувати товарами та здійснювати онлайн-продажі. У рамках роботи спроєктовано архітектуру клієнтської та серверної частин застосунку, створено зручну та ефективну базу даних для зберігання інформації про користувачів, товари, замовлення та доставки.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати існуючі рішення у сфері електронної комерції та маркетплейсів;
- розробити архітектуру вебзастосунку з поділом на клієнтську та серверну частини;
- реалізувати систему реєстрації та авторизації користувачів;
- створити інтерфейс для перегляду каталогу товарів та пошуку за категоріями;
- розробити функціонал кошика та оформлення замовлень;
- впровадити адміністративний модуль для управління категоріями, товарами та замовленнями;
- забезпечити збереження та обробку даних за допомогою реляційної або документної бази даних;
- реалізувати комп’ютерну модель (програмну реалізацію) системи у вигляді повноцінного вебзастосунку.

2 АРХІТЕКТУРА ЗАСТОСУНКУ ТА ПРОГРАМНІ ЗАСОБИ

2.1 Архітектура застосунку

Одним із найважливіших етапів у розробці сучасних інформаційних систем, зокрема вебзастосунків у сфері електронної комерції, є вибір та проєктування архітектури програмного забезпечення. Архітектура визначає принципи організації системи, її структурну побудову, спосіб комунікації між її компонентами, а також рівень масштабованості, надійності, супроводжуваності та безпеки. У контексті розробки маркетплейсу, який має об’єднувати численні функціональні блоки – обробку замовлень, управління товарами, реєстрацію користувачів, роботу з базами даних, платіжні модулі тощо – архітектурне рішення безпосередньо впливає на технічний успіх усього проєкту.

На сьогодні найбільш поширеними архітектурними підходами до побудови серверної частини вебзастосунків є монолітна архітектура та мікросервісна архітектура. Кожен з них має свої сильні сторони та обмеження, а вибір між ними залежить від масштабів проєкту, бізнес-вимог, доступних ресурсів, а також технічної компетенції команди розробників.

Монолітна архітектура передбачає створення застосунку як єдиного, нероздільного програмного модуля. Усі функції системи – обробка запитів, логіка бізнес-процесів, робота з базами даних, авторизація та аутентифікація, API, обслуговування інтерфейсу – розміщуються в одному кодовому репозиторії, розгортаються разом і працюють як єдине ціле.

У межах кваліфікаційної роботи обрана монолітна архітектура на базі фреймворку NestJS, який працює на платформі Node.js з адаптацією до Fastify – високопродуктивного HTTP-сервера. Такий вибір обумовлено низкою причин:

- швидкість розробки: у моноліті всі компоненти доступні в одному проєкті, що значно спрощує налагодження, виклик функцій між модулями, повторне використання спільного коду та реалізацію перших версій системи;
- менша складність інфраструктури: один екземпляр застосунку, одна база коду, один процес деплойменту – усе це значно знижує поріг входу та дозволяє зосередитися на реалізації бізнес-логіки, а не на налаштуванні контейнерів, оркестраторів і брокерів повідомлень;
- потужна модульна система NestJS: хоча проєкт монолітний, NestJS дозволяє логічно ізолювати функціональність у модулі: наприклад, UsersModule, ProductsModule, OrdersModule, StoresModule. Такий підхід дозволяє утримувати архітектуру чистою та зрозумілою, навіть за зростання кількості функцій;
- простота розгортання: для MVP-версії маркетплейсу важливо мати можливість швидко розгорнути застосунок на одному сервері або в одному контейнері, без необхідності налаштування складної CI/CD-інфраструктури.

При цьому, як показує практика, монолітна архітектура має і свої обмеження:

- зі збільшенням коду зростає складність внесення змін – навіть локальна зміна може вплинути на інші модулі;
- при зростанні навантаження виникає потреба в масштабуванні всієї системи, навіть якщо навантаження йде лише на окремий модуль (наприклад, каталог товарів);
- залежність модулів між собою може призвести до «спагетті-коду», якщо не дотримуватися принципів SOLID, DI та модульної декомпозиції.

Мікросервісна архітектура є протилежністю моноліту. У ній застосунок розбивається на низку незалежних сервісів, кожен з яких реалізує окрему бізнес-функцію (наприклад, управління товарами, обробка платежів, система аналітики, обробка зображень тощо). Кожен сервіс має власну базу даних або сховище, API, може бути написаний на іншій мові або фреймворку та масштабуватися окремо від інших.

Основні переваги мікросервісної архітектури:

- масштабованість: сервіс, що найбільше навантажений, масштабується незалежно від інших;
- ізоляція збоїв: падіння одного сервісу не призводить до повного краху системи;
- незалежний деплоймент: оновлення одного мікросервісу не зачіпає інші;
- гнучкість: команда може обрати найкращі технології для кожного сервісу (наприклад, використати Python для сервісу рекомендацій, Node.js для API, Go для обробки платежів).

Проте впровадження мікросервісної архітектури потребує високої зрілості команди та складної інфраструктури:

- потрібен сервісний реєстр, брокер повідомлень, балансувальник навантаження;
- складніше налаштувати безпечну взаємодію між сервісами;
- виникає проблема консистентності даних між різними базами;
- суттєво зростає вартість DevOps-обслуговування (CI/CD, моніторинг, логування, трасування).

Порівняння монолітної та мікросервісної архітектури представлено на рисунку 2.1.

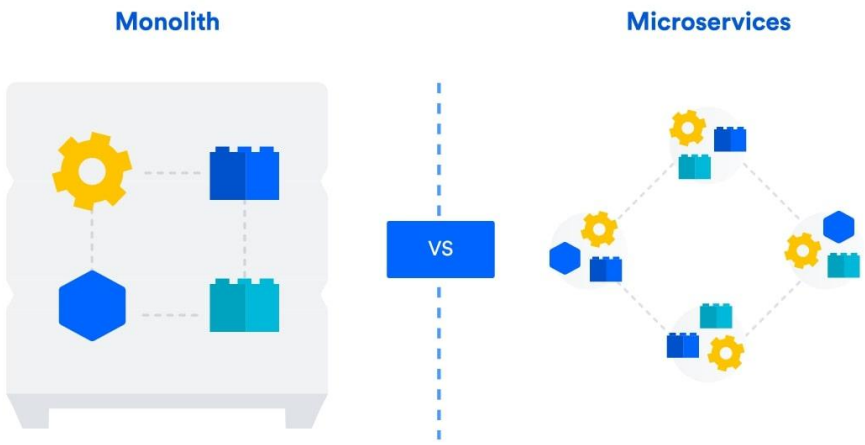

Рисунок 2.1 – Моноліт та мікросервіси

Саме тому для проєктів типу MVP маркетплейсу або перших версій з помірним навантаженням доцільно починати з моноліту, а вже згодом – при збільшенні команди, трафіку, складності – виділяти сервіси поступово, мігруючи архітектуру до мікросервісної.

У межах кваліфікаційної роботи маркетплейс реалізовано за допомогою монолітної архітектури. Вибір даного підходу був зумовлений відносною простотою розгортання, управління та оновлення застосунку на ранніх етапах проєкту. На відміну від мікросервісної архітектури, моноліт дозволяє уникнути додаткових витрат на налаштування сервісної взаємодії та інфраструктури.

Успішне проєктування архітектури інформаційних систем базується на принципах масштабованості, надійності та підтримуваності, що висвітлюються у класичних роботах з архітектури програмного забезпечення [11, 12]. Основою для вибору архітектурного рішення слугували принципи чіткої модульності, які спрощують подальшу підтримку та масштабування застосунку. При побудові архітектури застосовувалися сучасні патерни організації серверних систем, що дозволяють забезпечити масштабованість та зручність обслуговування великомасштабних застосунків [13].

Застосунок побудований відповідно до принципів REST-архітектури, яка передбачає використання стандартних HTTP-методів для взаємодії клієнта і сервера. Розробка RESTful API базується на принципах архітектури мережних застосунків, сформульованих у роботі Р. Філдінга [14]. Це забезпечує високу сумісність між різними компонентами системи та зручність у розширенні функціоналу [15].

2.2 Мови програмування

У сучасній цифровій епосі розробка й обслуговування вебсторінок стала невід’ємною складовою діяльності компаній та організацій. Основним

інструментом для створення структури вебконтенту є мова розмітки HTML, що визначає логічну організацію даних і забезпечує взаємодію з користувачами. HTML виступає базовою технологією веброзробки, яка дає змогу реалізовувати як прості статичні сайти, так і складні інтерактивні застосунки. Завдяки HTML можна інтегрувати текст, зображення, відео, форми зворотнього зв'язку та інші елементи, що робить цю мову універсальним інструментом для розробників.

HTML залишається основною мовою для побудови вебсторінок і вебзастосунків, виконуючи роль опису структури інформації через спеціальні теги, які підказують браузерам, як саме відображати вміст. Від моменту свого виникнення HTML став наріжним каменем світового вебпростору, забезпечуючи базову платформу для розміщення контенту в Інтернеті.

Однак для ефективної взаємодії з користувачами та створення привабливого зовнішнього вигляду вебсторінок важливу роль відіграє мова CSS. Вона відповідає за оформлення елементів HTML, дозволяючи контролювати кольори, шрифти, макети, розміщення блоків та анімації. Завдяки CSS розробники можуть створювати не лише статичні сторінки, а й складні адаптивні вебзастосунки із сучасним дизайном. Вона стала невід'ємною частиною веброзробки, розширюючи можливості з візуальної презентації та підвищуючи рівень користувацького досвіду.

CSS використовується для призначення стилів HTML-елементам, забезпечуючи гнучке керування візуальними аспектами вебінтерфейсу. Від часу свого впровадження CSS суттєво змінив підхід до розробки вебсайтів, дозволивши створювати естетично привабливі, зручні та адаптивні інтерфейси, що підвищують залученість користувачів.

Сучасний веброзробник стикається з великою кількістю різноманітних завдань: від створення розважальних порталів до розробки серйозних корпоративних систем, що потребують високого рівня надійності та безпеки. Для реалізації таких проєктів необхідно правильно обирати інструменти: мови програмування, фреймворки та бібліотеки. Кожна мова має свої

переваги в залежності від конкретних вимог проєкту, а вибір часто базується на знаннях і досвіді розробника.

Мова програмування – це засіб, який дозволяє людині давати інструкції комп'ютеру у формі, яку машина може зрозуміти і виконати. Вона створена для того, щоб описувати обчислення, логіку і послідовність дій у чіткій, однозначній формі.

Усі мови вебпрограмування умовно поділяють на дві великі групи: клієнтські та серверні. Клієнтські мови орієнтовані на створення програм, які виконуються безпосередньо у веббраузері користувача. Це означає, що код, написаний клієнтською мовою, завантажується з вебсервера на пристрій користувача і обробляється локально, у його браузері. Таким чином, клієнтські мови відповідають за візуальне оформлення сторінки, за інтерактивність, а також за швидку реакцію на дії користувача без необхідності постійного звернення до сервера. Прикладом такої мови є JavaScript. Вони визначають, як виглядає вебсторінка, як вона поводиться при взаємодії з користувачем та які анімації або переходи на ній відбуваються.

Серверні мови вебпрограмування, навпаки, працюють на стороні сервера. Вони застосовуються для обробки запитів користувача, взаємодії з базами даних, створення динамічного контенту та забезпечення логіки роботи вебзастосунків. Коли користувач, наприклад, надсилає форму реєстрації, серверна програма приймає дані, перевіряє їх, зберігає в базі даних або виконує інші необхідні дії, після чого може відправити результат назад у вигляді нової сторінки або повідомлення. Популярні серверні мови – це C#, Java, PHP, Python (фреймворки Django або Flask) та JavaScript (Node.js).

Найпоширенішою клієнтською мовою є JavaScript, оскільки саме вона виконується у фронтовій частині всіх сучасних браузерів і є стандартом для взаємодії з DOM-структурою сторінки. Вона дозволяє змінювати вміст вебсторінки в реальному часі без повного перезавантаження, забезпечуючи

плавну та інтерактивну взаємодію з користувачем. За допомогою JavaScript можна змінювати вигляд елементів, реагувати на події, створювати анімації, перевіряти форми, реалізовувати асинхронні запити до сервера, а також будувати повноцінні односторінкові застосунки (SPA).

Простота синтаксису, низький поріг входу та величезна екосистема бібліотек і фреймворків зробили JavaScript надзвичайно популярною мовою як серед початківців, так і серед професійних розробників. Крім веброзробки, ця мова активно використовується і в інших сферах: створення мобільних застосунків, ігор, а також у серверній розробці через платформу Node.js.

На 2025 рік, за даними порталу DOU [16], які показані на рисунку 2.2, найпоширенішою є надбудова над JavaScript – TypeScript. Вона дозволяє писати типізований код, що особливо важливо при розробці великих і довготривалих проєктів у команді.

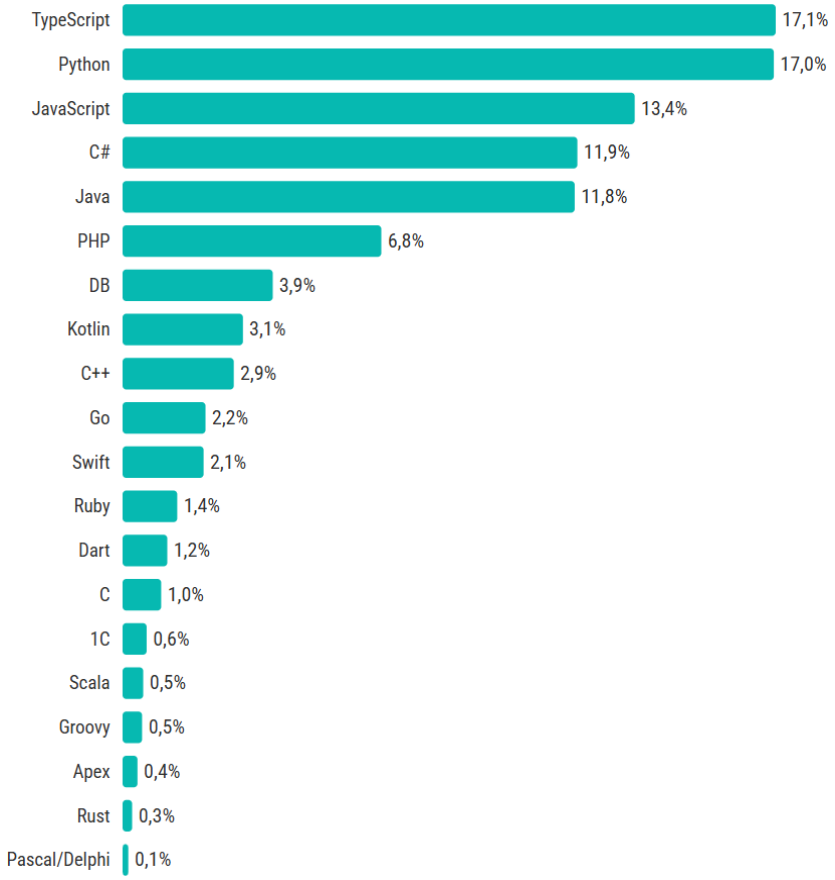

Рисунок 2.2 – Відсоток використання мов програмування в комерційній розробці

TypeScript забезпечує статичну перевірку типів, покращує автодоповнення в IDE, дозволяє створювати більш зрозумілий і підтримуваний код, а також запобігає багатьом поширеним помилкам на етапі компіляції. Завдяки цим перевагам, TypeScript активно використовується як у фронтенді, так і у бекенді.

У моєму проєкті як клієнтська, так і серверна частини реалізовані з використанням TypeScript, що дозволило забезпечити єдину типізацію, зручну інтеграцію між модулями, а також покращити масштабованість і стабільність застосунку

2.3 Фреймворки для розробки серверної частини

Фреймворк можна розглядати як динамічну бібліотеку для певної мови програмування, яка об’єднує в собі набір базових модулів і готових рішень. Його основною метою є полегшення процесу розробки, адже замість створення функціональності «з нуля» розробники можуть використовувати вже підготовлені шаблони та інструменти.

У контексті Node.js фреймворки відіграють важливу роль у розробці програмного забезпечення, сприяючи прискоренню створення застосунків, підвищенню їхньої безпеки та забезпеченню можливості масштабування системи.

Серед основних переваг використання фреймворків для Node.js можна виділити:

– підвищення продуктивності: завдяки готовим рішенням для стандартних завдань розробники можуть зосередитися на унікальних аспектах проєкту, що пришвидшує розробку;

– масштабованість: фреймворки полегшують розширення функціоналу застосунків і забезпечують їх ефективну роботу при збільшенні навантаження;

– зручність: інструменти, які надають фреймворки, оптимізують процес створення та обслуговування вебзастосунків;

– спрощення розробки: завдяки використанню готових компонентів і шаблонних рішень розробники можуть швидше створювати повноцінні системи;

– підвищення безпеки: багато сучасних фреймворків включають механізми захисту від типових вразливостей ще на етапі розробки.

Серед найпопулярніших фреймворків для Node.js слід зазначити:

– Express – легковажний та гнучкий фреймворк, який став своєрідним стандартом для створення серверних застосунків і API в екосистемі Node.js. Він дозволяє швидко й ефективно розробляти вебсервери;

– Koa – сучасний фреймворк, заснований на концепції промісів, який полегшує асинхронне програмування та пропонує спрощений підхід у порівнянні з Express.js;

– Fastify – високопродуктивний фреймворк для Node.js, орієнтований на створення швидких і масштабованих серверних застосунків. Fastify забезпечує високу швидкість обробки запитів завдяки оптимізованій архітектурі, підтримує вбудовану валідацію схем, автоматичну генерацію документації API та має потужну систему плагінів, що полегшує розширення функціональності застосунків.

Результати тестування продуктивності фреймворків (табл. 2.1) [17].

Таблиця 2.1 – продуктивність фреймворків

Фреймворк	**Запитів/сек**	**Відносна продуктивність (%)**
Fastify	47032	100%
Koa	37044	78,8%

Продовження таблиці 2.1

Фреймворк	**Запитів/сек**	**Відносна продуктивність (%)**
Restify	34754	73,9%
Hapi	31933	67,9%
Express	9920	21,1%

Окрім традиційних фреймворків, у Node.js-екосистемі існують також так звані метафреймворки, які будуються на базі інших фреймворків, один з яких це NestJS.

NestJS – це прогресивний метафреймворк для Node.js, який будується на базі Express.js або Fastify (на вибір) і використовує переваги мови TypeScript. NestJS пропонує структуру, натхненну архітектурними патернами корпоративного програмування, такими як MVC (Model-View-Controller) [18] та DI (Dependency Injection). Його модульна система дозволяє розділяти застосунок на незалежні частини, що спрощує розробку великих і масштабованих проєктів. Завдяки широкій екосистемі модулів, інтеграції з базами даних, можливостям побудови мікросервісів і підтримці WebSockets, NestJS є ідеальним вибором для побудови комплексних серверних застосунків.

Таким чином, використання фреймворків та метафреймворків у розробці на Node.js дозволяє значно оптимізувати процес створення застосунків, забезпечуючи високу якість, безпеку, масштабованість і спрощене обслуговування програмних рішень, що робить їх незамінною частиною сучасної веброзробки.

2.4 Фреймворки для розробки клієнтської частини

Фреймворки для фронтенд-розробки є спеціалізованими наборами інструментів та бібліотек, які допомагають розробникам створювати

інтерактивні, зручні та продуктивні інтерфейси користувача. Вони забезпечують структуровану архітектуру проєкту, стандартизовані рішення для управління станом застосунку, маршрутизації, взаємодії з API та обробки подій. Застосування фреймворків дозволяє суттєво скоротити час розробки, підвищити якість коду та забезпечити його масштабованість і підтримуваність.

У сучасній веброзробці фреймворки для клієнтської частини допомагають реалізовувати динамічні інтерфейси з використанням компонентного підходу, що сприяє підвищенню гнучкості та повторного використання коду. Завдяки широкому екосистемному оточенню фреймворки підтримують інтеграцію з численними бібліотеками для управління станом, анімаціями, побудовою форм та оптимізації продуктивності.

Фронтенд-фреймворки дають змогу реалізовувати односторінкові вебзастосунки (SPA), де навігація між різними розділами здійснюється без повного перезавантаження сторінки (рис. 2.3). Завдяки цьому підходу користувачі отримують плавний і швидкий досвід взаємодії з застосунком, що значно підвищує зручність використання.

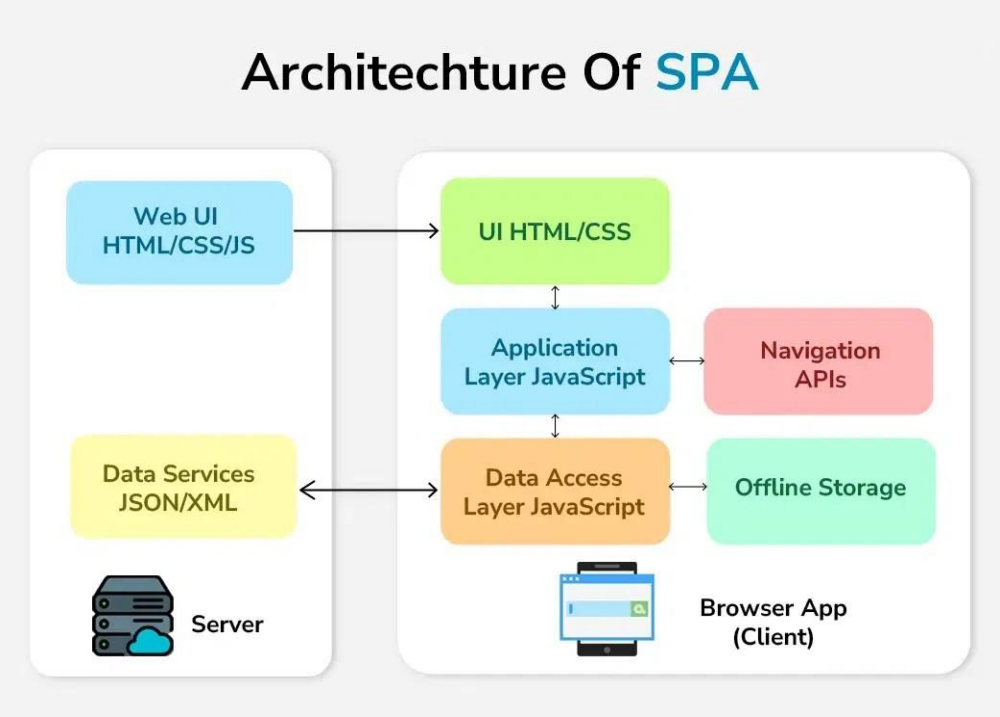

Рисунок 2.3 – Архітектура односторінкового вебзастосунку

Серед найпопулярніших фреймворків для розробки клієнтської частини слід відзначити:

– React – бібліотека для створення інтерфейсів користувача, яка часто розглядається як фреймворк завдяки своїй великій екосистемі. React дозволяє розробляти компоненти, які ефективно оновлюються та відображаються завдяки концепції віртуального DOM [19]. Гнучкість React сприяє його широкому використанню у створенні як простих сайтів, так і складних вебзастосунків;

– Vue – прогресивний фреймворк, орієнтований на простоту інтеграції та швидкий старт. Vue поєднує найкращі риси React та Angular, пропонуючи декларативний синтаксис, двостороннє зв’язування даних та легку модульність, що робить його ідеальним вибором як для малих, так і для великих проєктів;

– Angular – повноцінний фреймворк від компанії Google, який пропонує все необхідне для розробки масштабованих вебзастосунків: від системи компонентів до вбудованої маршрутизації та механізмів для роботи з формами й HTTP-запитами.

Порівняння основних фронтенд-фреймворків наведено у таблиці 2.2.

Таблиця 2.2 – порівння фронтенд фреймворків

Характеристика	**React**	**Angular**	**Vue.js**
Розробник	Meta (Facebook)	Google	Evan You
Розмір бандлу	Середній	Великий	Малий
Зв’язування даних	Одностороннє	Двостороннє	Двостороннє
Віртуальний DOM	Так	Ні	Так
Використання	SPA, динамічні вебзастосунки, мобільні застосунки	Корпоративні застосунки, складні SPA	Прототипи, поступова інтеграція

Також, як і з фрейворками для розробки серверної частинки, варті уваги метафреймворки для клієнту , які базуються на популярних рішеннях і пропонують додаткову функціональність для побудови складних застосунків:

– Next – це метафреймворк, який розширює можливості React. Next.js забезпечує серверний рендеринг (SSR), статичну генерацію сторінок (SSG) та покращене SEO без необхідності складної ручної конфігурації. Він також пропонує автоматичну маршрутизацію, вбудовану підтримку API-роутів і оптимізацію продуктивності, що робить його чудовим вибором для створення сучасних вебзастосунків;

– Nuxt – рішення, створене на базі Vue.js. Nuxt спрощує процес розробки універсальних (із серверним рендерингом) або статично згенерованих вебзастосунків, автоматизуючи багато складних процесів конфігурації. Завдяки вбудованій маршрутизації, генерації SEO-оптимізованих сторінок, можливості розширення за допомогою модулів та чудовій підтримці інтеграцій із серверними API, Nuxt.js є відмінним вибором для побудови сучасних масштабованих клієнтських рішень.

Отже, використання фронтенд-фреймворків та метафреймворків дозволяє створювати високопродуктивні, масштабовані та зручні для користувачів інтерфейси, що є невід’ємною частиною розробки сучасних вебзастосунків.

2.5 Бази даних

У сучасному світі, який розвивається від індустріальної до інформаційної епохи, автоматизовані системи зберігання інформації набувають ключового значення. Такі системи можуть мати значні обсяги та включати різні типи даних, необхідних для функціонування підприємств.

Інформаційний банк – це спеціалізована система, призначена для централізованого накопичення, актуалізації та пошуку інформації за

запитами користувачів. Вона складається з технічних засобів, програмного забезпечення та фахівців, які забезпечують її стабільну роботу. Такий банк може об'єднувати одну або кілька баз даних, систему керування ними та персонал, що відповідає за їх експлуатацію.

База даних – це організована структура, призначена для зберігання, управління та швидкого доступу до даних. Вона дозволяє ефективно працювати з великими обсягами інформації, забезпечуючи її цілісність, безпеку та можливість обміну між різними користувачами або програмами.

Одним із важливих аспектів вибору бази даних для конкретного проєкту є розуміння ключових відмінностей між двома основними підходами до організації даних: реляційними (SQL) та нереляційними (NoSQL) базами даних.

Реляційні бази даних базуються на суворій структурованій моделі даних. Інформація зберігається у вигляді таблиць із визначеними стовпцями та типами даних, що дозволяє забезпечити високу цілісність даних за допомогою механізмів обмежень (constraints), транзакцій та нормалізації. SQL-бази є ідеальним вибором для систем, де критично важливо дотримуватися суворої структури даних, наприклад, у фінансових застосунках, CRM-системах або великих e-commerce рішеннях.

У свою чергу, NoSQL бази даних надають більше гнучкості у зберіганні даних. Вони орієнтовані на роботу з великими обсягами нереляційних або слабо структурованих даних, що дає змогу швидше масштабувати систему горизонтально. NoSQL бази діляться на кілька підтипів за моделлю зберігання даних: документні (MongoDB), графові (Neo4j), бази типу «ключ-значення» (Redis) та колоночні (Cassandra). Вони особливо добре підходять для обробки великих потоків даних реального часу, для зберігання JSON-подібних документів, роботи зі складними об'єктами або побудови систем рекомендацій.

Класифікацію баз даних за типами наведено на рисунку 2.4.

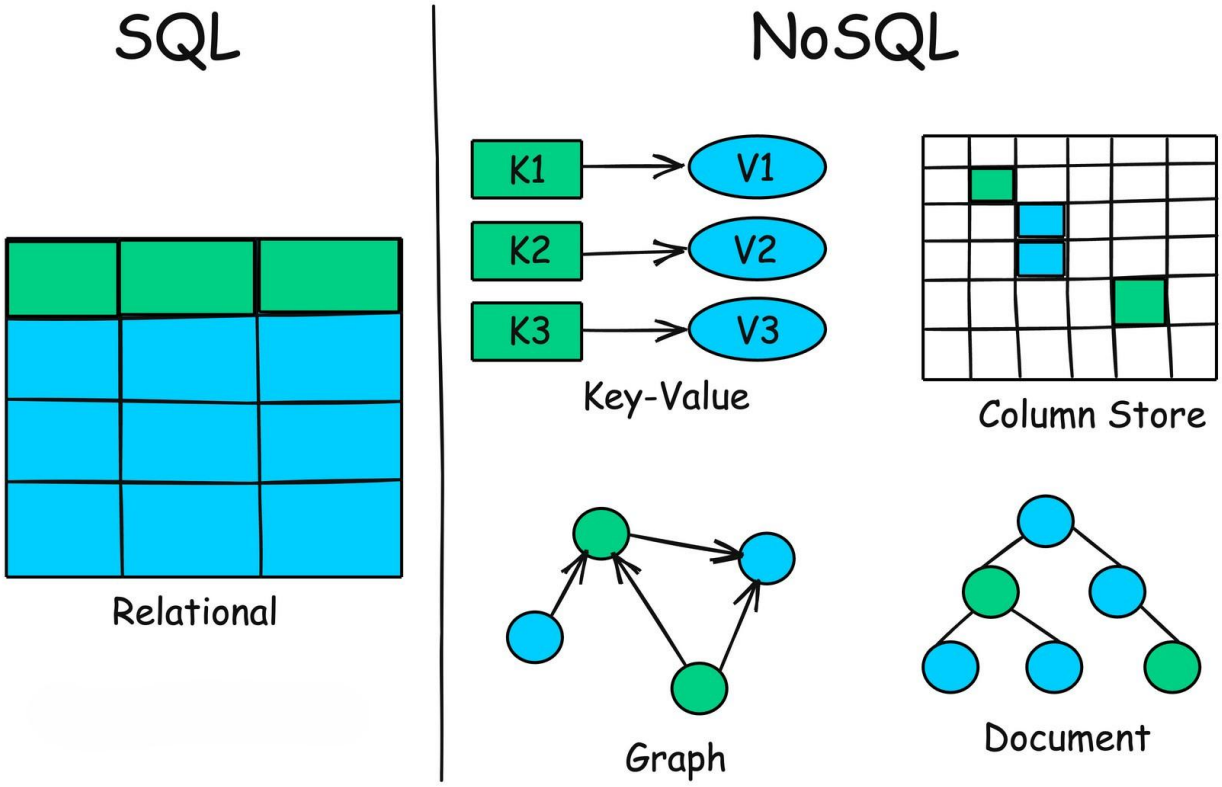

Рисунок 2.4 – Типи баз даних

Найпопулярніші СКБД за рейтингом DB-Engines [20], який оцінює популярність систем на основі кількості згадок у пошукових системах, інтересу користувачів (Google Trends), обговорень на технічних форумах, вакансій, профілів у LinkedIn та активності в соціальних мережах. Популярність визначається шляхом стандартизації і середнього обчислення цих параметрів, де результат є відносним і порівняльним між системами (табл. 2.3).

Таблиця 2.3 – рейтинг СКБД

Місце	**Назва СКБД**	**Модель бази даних**
1	Oracle	Реляційна, мультимодельна
2	MySQL	Реляційна, мультимодельна
3	Microsoft SQL Server	Реляційна, мультимодельна
4	PostgreSQL	Реляційна, мультимодельна

Продовження таблиці 2.3

Місце	Назва СКБД	Модель бази даних
5	MongoDB	Документна, мультимодельна
6	Snowflake	Реляційна
7	Redis	Ключ-значення, мультимодельна
8	Elasticsearch	Пошукова, мультимодельна
9	IBM Db2	Реляційна, мультимодельна
10	SQLite	Реляційна

Oracle – це комерційна реляційна СКБД, яка також підтримує мультимодельні можливості, зокрема роботу з JSON, XML, графовими структурами даних. Вона орієнтована на великі корпоративні системи, де критично важливі надійність, висока доступність та розвинені механізми безпеки. Oracle пропонує розширену підтримку транзакцій, складні системи резервного копіювання та відновлення даних, механізми реплікації і шардінгу.

MySQL – одна з найвідоміших систем з відкритим вихідним кодом, яка використовується для вебпроєктів, електронної комерції та SaaS-застосунків. Завдяки своїй простоті та високій продуктивності MySQL залишається стандартом де-факто для багатьох вебплатформ. З часом MySQL також отримала підтримку JSON-типів даних та функцій для роботи з нереляційними структурами.

SQL Server від Microsoft активно використовується в корпоративних застосунках. Він забезпечує високу інтеграцію з іншими продуктами Microsoft (зокрема Azure) і пропонує розширені аналітичні можливості, інструменти для обробки великих даних і побудову data warehouse-рішень.

PostgreSQL – потужна об’єктно-реляційна система з відкритим вихідним кодом. Вона відома своєю розширюваністю, підтримкою складних типів даних (множинні типи, JSONB), високою відповідністю стандартам SQL та потужними можливостями створення власних функцій і тригерів.

PostgreSQL активно використовується в проєктах, де потрібна надійність, масштабованість і складна бізнес-логіка.

MongoDB є найпопулярнішою документною NoSQL базою даних. Вона дозволяє зберігати дані у форматі BSON (розширення JSON) і забезпечує гнучку модель даних. MongoDB легко масштабується горизонтально завдяки вбудованій підтримці шардінгу, що робить її ідеальним вибором для проєктів із великими обсягами нереляційних даних, наприклад, соціальних мереж або платформ електронної комерції.

Redis – це база даних типу «ключ-значення», яка зберігає дані в оперативній пам’яті, що забезпечує надзвичайно швидкий доступ до інформації. Redis часто використовується для кешування, сесійного зберігання, побудови черг завдань та реального часу аналітики.

Elasticsearch є потужною пошуковою системою, яка також виконує роль бази даних для великих обсягів неструктурованих текстових даних. Вона широко використовується для побудови механізмів пошуку, систем логування (наприклад, у складі ELK-стека) та аналітики даних у реальному часі.

У процесі вибору бази даних для розробки вебмаркетплейсу варто орієнтуватися на такі критерії, як масштабованість, підтримка складних зв’язків між даними, продуктивність, можливості горизонтального масштабування та вимоги до гнучкості структури. З огляду на ці параметри, PostgreSQL і MongoDB часто є найкращими варіантами для сучасних вебзастосунків, що орієнтуються на швидке розширення функціоналу та роботу з різнорідними даними.

3 РОЗРОБКА ВЕБМАРКЕТПЛЕЙСУ ДЛЯ ЕЛЕКТРОННОЇ КОМЕРЦІЇ З МОЖЛИВІСТЮ СТВОРЕННЯ КОРИСТУВАЦЬКИХ МАГАЗИНІВ

3.1 Обґрунтування вибору середовища та інструментів програмної реалізації

Під час розробки повнофункціонального вебмаркетплейсу для електронної комерції особливу увагу було приділено вибору середовища програмної реалізації – як для серверної, так і для клієнтської частини застосунку. Від правильного вибору інструментів розробки, фреймворків, мов програмування, а також середовища розробки залежить не лише продуктивність та масштабованість застосунку, але й зручність роботи, швидкість розробки та стабільність функціоналу.

У даному проєкті було використано інтегроване середовище розробки JetBrains WebStorm, яке на сьогодні вважається одним із найпотужніших та найзручніших середовищ для роботи з JavaScript/TypeScript, React, Node.js, а також сучасними фреймворками (NestJS, Next.js, Prisma, Redux тощо). WebStorm забезпечує підтримку всіх сучасних стандартів та технологій веброзробки, має гнучку систему плагінів, розширень, а також вбудовані засоби налагодження, контролю версій, форматування, рефакторингу та тестування.

Основні переваги WebStorm, що обумовили його вибір:

- автоматична підказка коду та автодоповнення, що значно підвищує швидкість написання коду та зменшує кількість синтаксичних помилок;
- інтеграція з системами контролю версій (Git, GitHub), що дає змогу вести повноцінну розробку в команді;
- гнучке налаштування середовища, включно з ESLint, Prettier, TypeScript-компілятором і багатьма іншими інструментами, що використовуються в проєкті;

- можливість одночасної роботи з кількома package.json (frontend і backend частини), що є критично важливим при роботі з монорепозиторіями;
- потужна підтримка налагодження (debugging) як у середовищі Node.js, так і в браузері (через Chrome Debugger).

У якості середовища для реалізації серверної частини було обрано NestJS – прогресивний TypeScript-метафреймворк, побудований поверх Node.js із використанням архітектурних патернів корпоративного рівня (Dependency Injection, модульність, Middleware, Pipes, Guards тощо) [21]. У якості HTTP-сервера використано Fastify – високопродуктивну альтернативу Express.js, яка за результатами тестів демонструє майже п'ятикратну перевагу в обробці запитів.

Додатково серверна частина застосунку базується на наступних інструментах:

- TypeScript – основна мова програмування, яка забезпечує безпеку типів, покращене автодоповнення, зменшення кількості помилок та кращу підтримку у IDE;
- Prisma ORM – сучасний ORM-інструмент для роботи з базами даних, що дозволяє заощаджувати час на написання SQL-запитів, надає типізовану модель бази даних, інтегрується з TypeScript, підтримує генерацію міграцій та seed-даних;
- JWT та Passport – для реалізації безпечної автентифікації користувачів;
- Class-validator та class-transformer – для перевірки даних при обробці запитів.

Конфігурація серверної частини демонструє глибоку інтеграцію з інструментами автоматизації та тестування: ESLint, Prettier, Jest (з підтримкою e2e), ts-node, tsconfig-paths, seed-скрипти для заповнення бази тощо. Наявність таких скриптів, як start:dev, start:debug, test:cov, lint, format, свідчить про дотримання сучасних DevOps-практик та CI/CD-підходів, що дозволяє легко масштабувати проєкт та впроваджувати тестування.

Фронтенд було реалізовано з використанням Next.js – сучасного фреймворку, що поєднує в собі можливості серверного рендерингу (SSR), статичної генерації (SSG) та побудови SPA. У зв'язку з підтримкою App Router, React 19 та Turbopack, цей стек дозволив створити високопродуктивний, масштабований та зручний для SEO вебінтерфейс.

Важливими компонентами фронтенду є:

- React – бібліотека для побудови інтерфейсу, яка забезпечує компонентну архітектуру та декларативний підхід;
- Redux Toolkit та React Query – для зберігання та обробки глобального стану застосунку, кешування та оптимізації запитів;
- TailwindCSS – сучасна утилітарна CSS-бібліотека для швидкої верстки з підтримкою адаптивності, темізації та анімацій;
- Zod, React Hook Form, Radix UI – для побудови форм, валідації даних та зручного інтерфейсу.

Використання axios забезпечує асинхронну взаємодію з сервером, а бібліотеки clsx, lucide-react, recharts покращують візуальну складову платформи: іконки, графіки, динамічні компоненти.

Клієнтська частина також реалізована на TypeScript, що забезпечує спільну типізацію з бекендом, зменшуючи кількість помилок під час інтеграції між клієнтом і сервером.

Обрана архітектура (NestJS + Prisma + Fastify на бекенді, Next.js + React + Tailwind на фронтенді) відповідає сучасним вимогам до продуктивності, безпеки та масштабованості вебзастосунків. Використання однієї мови програмування (TypeScript) на обох частинах системи дозволило забезпечити спільну логіку валідації, полегшити супровід та спростити командну роботу.

Завдяки обраним інструментам, обидві частини проєкту мають чітко структуровані скрипти для запуску, тестування, форматування, що дозволяє легко адаптуватися до CI/CD-інфраструктури в майбутньому. Крім того, вся система готова до контейнеризації, оскільки не містить платформозалежних залежностей.

3.2 Етапи програмної реалізації

Розробка вебмаркетплейсу є багатогранним процесом, який охоплює низку послідовних етапів. На кожному з них вирішуються завдання, пов’язані з архітектурою системи, розробкою інтерфейсів, логікою взаємодії, безпекою та забезпеченням масштабованості проєкту. У цьому розділі деталізовано основні етапи програмної реалізації, які були здійснені під час створення маркетплейсу.

3.2.1 Проєктування загальної архітектури

Для гарної програмної реалізації на потрібна продумана архітектури системи, що визначає загальний вигляд, взаємозв’язки та принципи взаємодії компонентів маркетплейсу.

В умовах сучасного веброзробництва стандартом є використання клієнт-серверної архітектури, що забезпечує чіткий поділ між логікою інтерфейсу користувача (frontend) і серверною логікою (backend). Такий підхід дозволяє:

- масштабувати кожну частину незалежно;
- забезпечити безпеку, приховуючи бізнес-логіку та дані на сервері;
- організувати гнучку інтеграцію з іншими сервісами;
- полегшити супровід і тестування системи.

Архітектура системи побудована за принципом розділення відповідальностей (Separation of Concerns). Усі запити користувача спочатку обробляються клієнтською частиною, яка формує відповідний HTTP-запит до серверної частини. Бекенд отримує запит, виконує обробку, за потреби звертається до бази даних, формує відповідь і повертає її клієнту через API (рис. 3.1). Така структура забезпечує гнучкість, масштабованість і легкість підтримки проєкту.

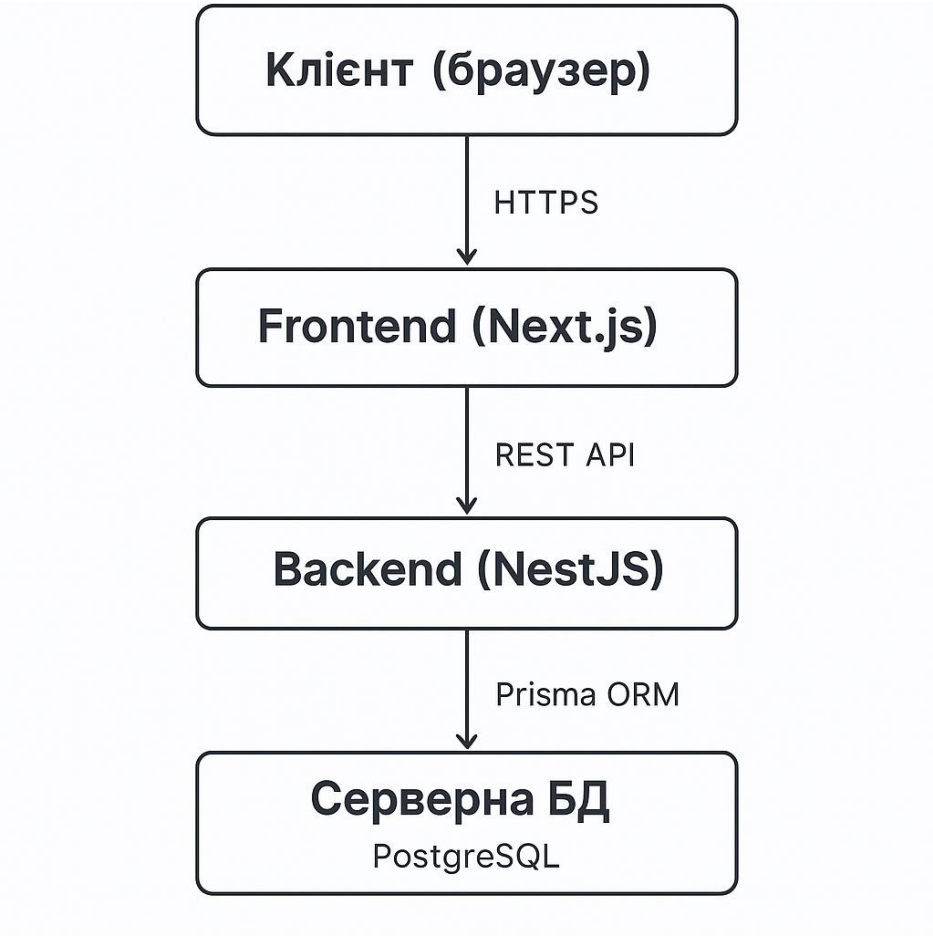

Рисунок 3.1 – Схематичне зображення загальної архітектури

Архітектура системи базується на сучасних підходах до розробки, які забезпечують її масштабованість, гнучкість, безпеку та стійкість до навантажень. Впроваджені принципи дозволяють ефективно керувати розвитком проєкту, підтримувати його стабільність і забезпечувати якісний користувацький досвід.

Основні принципи:

- модульність – кожен компонент системи має чітко визначену зону відповідальності. Це дозволяє спрощувати розробку, тестування та обслуговування коду;
- розширюваність – архітектура спроєктована так, щоб легко інтегрувати нову функціональність без необхідності змін у вже реалізованих модулях;

- безпека – усі дані обробляються виключно на серверній стороні. Обмін інформацією між клієнтом і сервером відбувається через захищені канали зв’язку (HTTPS);
- швидкодія – оптимізовано обробку запитів і взаємодію з базою даних. Застосовуються методи кешування та ефективні структури запитів;
- надійність – система здатна продовжувати роботу навіть у випадку відмови окремих компонентів, завдяки централізованій обробці помилок і механізмам відновлення.

Вся взаємодія між клієнтською та серверною частинами відбувається через REST API за допомогою HTTP-запитів (табл. 3.1). Користувач працює з інтерфейсом на фронтенді, який, у відповідь на дії (наприклад, реєстрація, створення магазину, оформлення замовлення), надсилає відповідні запити до серверної частини.

Сервер (бекенд) обробляє запит: перевіряє права доступу, валідує дані, взаємодіє з базою даних (PostgreSQL) та формує відповідь. Ця відповідь повертається на клієнт, де відображається результат дії.

Таблиця 3.1 – Приклади HTTP-запитів

Дія	**Ініціатор**	**Взаємодія**
Реєстрація	Користувач	Фронтенд → Бекенд (POST /register) → Запис у БД
Вхід	Користувач	Фронтенд → Бекенд (POST /login) → Перевірка в БД
Створення магазину	Продавець	Фронтенд → Бекенд (POST /stores) → Запис у БД
Додавання товару	Продавець	Фронтенд → Бекенд (POST /products) → Запис у БД
Оформлення замовлення	Покупець	Фронтенд → Бекенд (POST /orders/place) → Запис у БД

Таким чином, кожна частина системи виконує свою роль у рамках чітко визначених меж, дотримуючись принципу розділення відповідальностей і забезпечуючи цілісну, надійну роботу платформи.

3.2.2 Проєктування структури бази даних

Структура бази даних є критичним аспектом для будь-якого маркетплейсу, оскільки саме вона визначає ефективність зберігання, пошуку, оновлення й обробки інформації про користувачів, магазини, товари та замовлення. На цьому етапі було виконано глибоке моделювання предметної області та розроблено оптимізовану структуру даних, що відповідає функціональним вимогам системи.

У проєкті використовується реляційна база даних PostgreSQL, модель якої реалізовано з використанням ORM Prisma [22, 23]. Архітектура бази побудована відповідно до принципів нормалізації та підтримує зв’язки між користувачами, магазинами, товарами, замовленнями та іншими сутностями.

Основні таблиці (моделі):

- User – зберігає інформацію про користувачів платформи: електронна пошта, пароль, ім’я, зображення профілю. Користувач може створювати магазини, робити замовлення, писати відгуки та додавати товари у вибране;
- Store – представляє магазин, який належить певному користувачу. Містить назву, опис, а також зв’язки з товарами, категоріями, кольорами, відгуками та замовленнями;
- Product – описує товар із назвою, описом, ціною, зображеннями. Прив’язаний до магазину, може мати категорію, колір, і бути у відгуках, замовленнях та списках обраного;
- Category – категорії товарів. Прив’язані до конкретного магазину. Дозволяють структурувати асортимент продукції;

- Color – колір товару, також прив'язаний до магазину. Це дає змогу мати унікальні палітри для кожного продавця;
- Review – модель для зберігання відгуків користувачів. Може стосуватися як товару, так і магазину. Містить текст і числовий рейтинг;
- UserFavorite – проміжна таблиця, яка реалізує зв'язок «багато до багатьох» між користувачами та товарами, дозволяючи додавати продукцію до списку обраного;
- Order – замовлення користувачів. Містить статус (очікується, оплачено, неуспішно), список товарів, суму та зв'язок із користувачем;
- OrderItem – товари в замовленні. Зберігає кількість, ціну, зв'язки з товаром, магазином та замовленням.

Візуальна структура цих усих таблиць і їхніх зв'язків показана на рисунку 3.2.

Рисунок 3.2 – Схематичне зображення бази даних

Отже, на етапі проєктування структури бази даних було створено чітку модель даних із урахуванням всіх вимог маркетплейсу. Це забезпечило надійну основу для подальшої розробки серверної логіки та ефективної роботи всієї платформи.

3.2.3 Реалізація серверної логіки

Бекенд-частина маркетплейсу виконує роль ядра програмного забезпечення, де реалізується основна бізнес-логіка, обробка клієнтських запитів, взаємодія з базою даних, авторизація та валідація доступу. Метою даного етапу є побудова надійної, безпечної та масштабованої серверної інфраструктури, яка забезпечує повноцінну роботу функціоналу платформи.

Для реалізації серверної частини застосовано NestJS – прогресивний фреймворк для створення серверних застосунків на базі Node.js з підтримкою TypeScript. У якості HTTP-сервера використовується високопродуктивний адаптер Fastify [24], що забезпечує високу швидкість обробки запитів і низьке споживання ресурсів.

Доступ до бази даних реалізовано за допомогою Prisma ORM, що забезпечує типобезпечний, декларативний інтерфейс взаємодії з реляційними базами даних. Prisma підтримує складні SQL-запити, транзакції, зв’язки між таблицями, а також автоматичну генерацію клієнтського API на основі схеми бази даних. Такий підхід дозволяє значно скоротити кількість ручного коду та зменшити ймовірність помилок при взаємодії з базою.

Архітектура застосунку побудована відповідно до модульного підходу, характерного для NestJS. Кожна функціональна частина системи реалізована як окремий модуль, що відповідає принципам SOLID, зокрема принципам єдиної відповідальності та інверсії залежностей. Це забезпечує високу читабельність коду, спрощує тестування та розширення функціональності у майбутньому.

Кожна сутність доменної моделі – користувачі, магазини, товари, замовлення – представлена окремим модулем з повноцінною підтримкою CRUD-операцій. Для взаємодії з клієнтом усі маршрути реалізовано у форматі REST API відповідно до стандартів протоколу HTTP. Повний перелік основних маршрутів подано в таблиці 3.2.

Таблиця 3.2 – Основні маршрути

Метод	**Шлях**	**Призначення**
POST	/auth/register	Реєстрація нового користувача
POST	/auth/login	Аутентифікація користувача
GET	/stores/:id	Отримання магазину по id
POST	/stores	Створення нового магазину
GET	/products	Пошук і перегляд товарів
POST	/products/:storeId	Додавання товару до магазину
GET	/users/profile	Перегляд інформації користувача
POST	/orders/place	Створення нового замовлення

Механізм аутентифікації в клієнтській частині реалізовано за допомогою токенів у форматі JWT (JSON Web Token) [25]. Після успішної авторизації користувача на сервері генерується унікальний токен, який містить зашифровану інформацію про користувача (зокрема, ідентифікатор та термін дії). Цей токен повертається клієнту та зберігається у браузері в cookies – це дозволяє зберігати сесію між оновленнями сторінки та в межах кількох вкладок.

При кожному наступному запиті до захищених маршрутів API токен автоматично додається до заголовків (через механізми обробки axios), що дає змогу серверу перевірити справжність користувача та дозволити або заборонити доступ до відповідних ресурсів. Зберігання токена саме в cookies забезпечує додатковий рівень безпеки, оскільки він може бути позначений як HttpOnly та Secure, що зменшує ризики викрадення токена через XSS-атаки.

Контроль доступу до захищених маршрутів реалізовано за допомогою Guard-компонентів NestJS, які перевіряють наявність та дійсність JWT-токена в кожному запиті. У разі відсутності токена або його недійсності запит блокується з відповідним повідомленням про помилку. Такий підхід забезпечує базовий рівень безпеки та дозволяє відокремити відкриті ендпоїнти (наприклад, реєстрацію або авторизацію) від приватних, доступ до яких мають лише автентифіковані користувачі).

Для валідації вхідних даних використовуються вбудовані механізми NestJS у поєднанні з бібліотекою class-validator. Це забезпечує централізовану перевірку структури та типів даних у DTO (Data Transfer Object).

Візуальне представлення реалізації серверної частини представлене на рисунку 3.3.

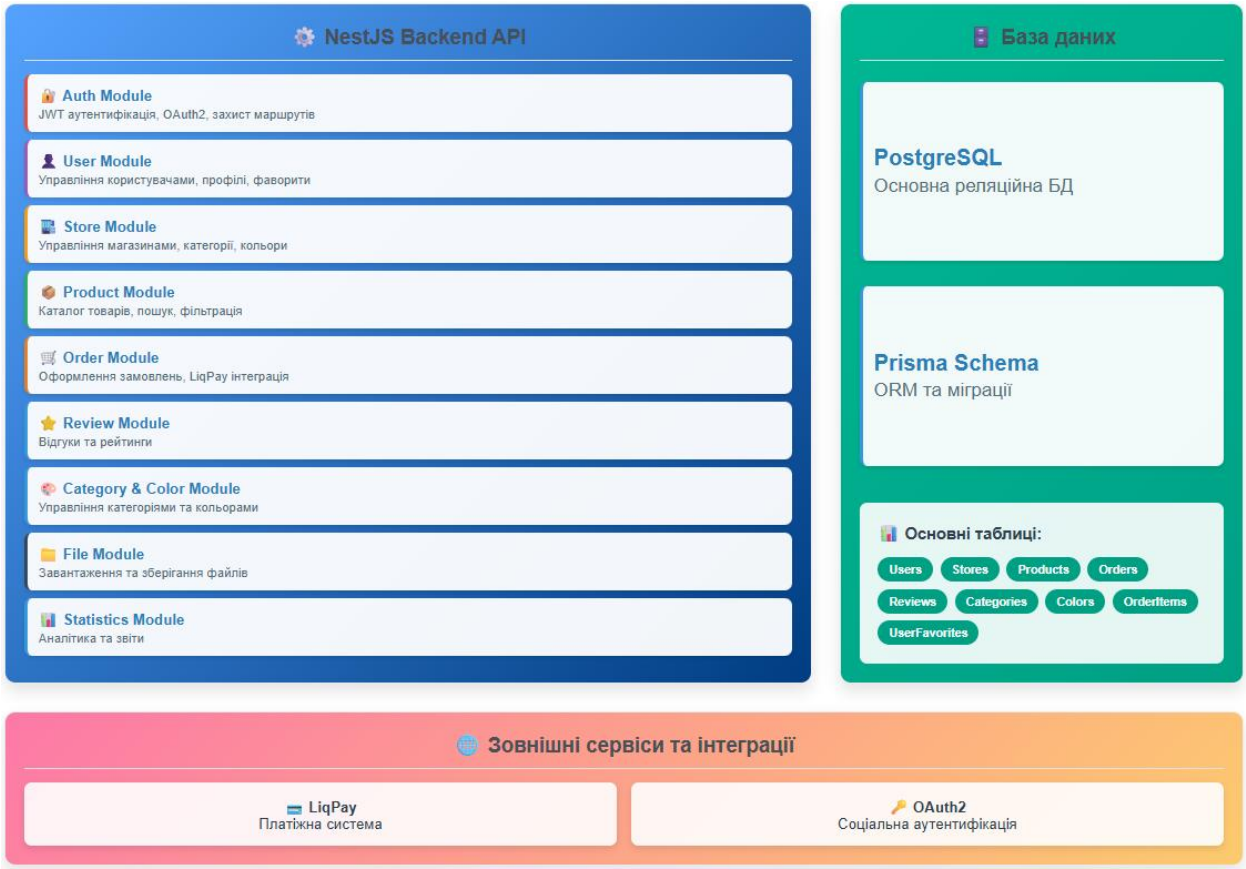

Рисунок 3.3 – Візуальне представлення серверної частини

Серверна частина платформи, як реалізована на основі сучасних технологій та обрана архітектура забезпечує гнучкість, безпеку, масштабованість та зручність розширення функціональності в майбутньому.

3.2.4 Реалізація клієнтської логіки

Фронтенд-частина маркетплейсу є основним інтерфейсом взаємодії користувачів із системою. Вона визначає перше враження про платформу, тому її якість, швидкодія, адаптивність та зручність використання мають вирішальне значення для успіху проєкту. Під час реалізації було створено сучасний, багатофункціональний інтерфейс, що враховує потреби різних типів користувачів – продавців, покупців, адміністраторів.

Фронтенд реалізовано на базі фреймворку Next.js [26], який поєднує переваги React з підтримкою серверного рендерингу та оптимізованої маршрутизації. Для типобезпеки використовується TypeScript, а стилізація здійснюється за допомогою SCSS та Tailwind CSS з плагіном tailwindcss-animate. Проєкт побудовано на компонентній архітектурі з чітким розділенням відповідальностей між модулями, що забезпечує легкість у підтримці та масштабуванні.

Інтерфейс створено з урахуванням сучасних принципів UI/UX-дизайну, включно з підтримкою адаптивності, темної теми, кросбраузерної сумісності та доступності. Компоненти побудовано з використанням бібліотеки Shadcn, який за основу бере Radix UI, що забезпечує стандартизовану поведінку та високу доступність модальних вікон, діалогів, селектів тощо.

Для створення інтерактивних ефектів застосовано clsx, tailwind-merge, а також анімації через плагін Tailwind. Сторінки оптимізовані під мобільні пристрої (mobile-first), із використанням CSS Flexbox та Grid Layout.

Маршрутизація реалізована через файлову систему Next.js. Для обмеження доступу до приватних сторінок застосовано захищені компоненти

з перевіркою автентифікації та стану користувача. Відображення інтерфейсу адаптується відповідно до типу користувача (покупець, продавець, адміністратор).

Глобальний стан застосунку реалізовано через Redux Toolkit [27], а для збереження міжсесійних даних використано redux-persist. Для асинхронних операцій і роботи з API впроваджено TanStack React Query [28], що дозволяє автоматично кешувати відповіді, повторно використовувати запити та синхронізувати стан із сервером.

Усі запити до бекенду виконуються через модуль services, побудований на базі axios. Для кожної бізнес-сутності створено окремий сервіс, що містить методи для CRUD-операцій. Це спрощує підтримку коду, забезпечує повторне використання логіки запитів та централізовану обробку помилок.

Для побудови форм застосовується React Hook Form, що забезпечує ефективну роботу з полями без зайвих ререндерів. Валідація форм здійснюється через бібліотеку Zod [29], яка дозволяє створювати типобезпечні схеми перевірки введених даних.

Інтерфейс активно використовує інтерактивні елементи, а також засоби інформування користувача:

- @dnd-kit – для drag-and-drop взаємодії (сортування товарів, зміна порядку);
- react-hot-toast – для toast-нотифікацій про дії користувача;
- recharts – для візуалізації статистичних даних (замовлення, відгуки);
- react-simple-star-rating – для системи оцінювання товарів;
- lucide-react – сучасна іконографіка;
- react-colorful – для вибору кольорів товарів;
- cmdk – для швидкої навігації та командного пошуку;
- js-cookie – для роботи з cookies (сесії, токени).

Використані технології та візуалізовану архітектуру клієнтської частини можна побачити на рисунку 3.4.

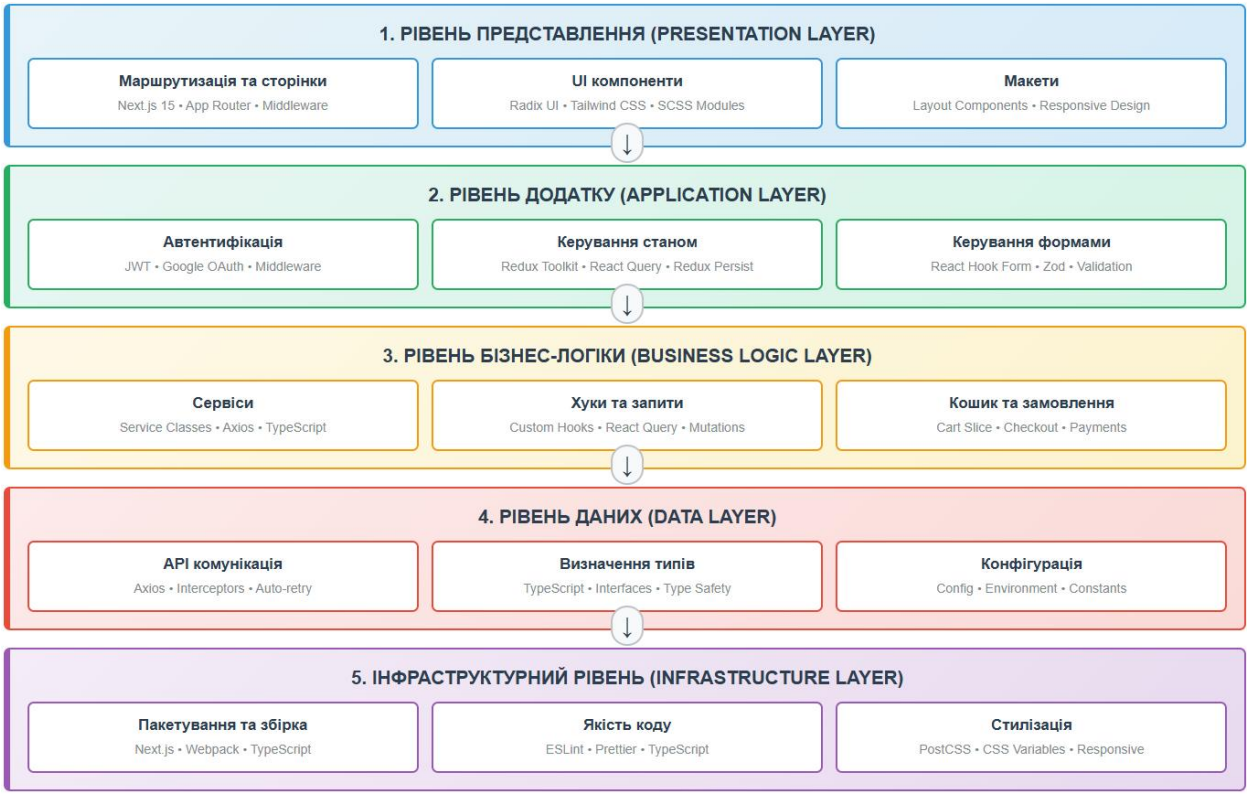

Рисунок 3.4 – Архітектура клієнтської частини

Клієнтська частина використовує сучасні веб-технології з акцентом на продуктивність, безпеку та масштабованість. Високий рівень типобезпеки, модульність архітектури, підтримка SSR/CSR через Next.js, а також інтеграція з API забезпечують стабільну роботу інтерфейсу та позитивний досвід взаємодії для всіх категорій користувачів платформи.

3.3 Тестування вебзастосунку

Для перевірки функціональності вебмаркетплейсу було проведено комплексне тестування, у межах якого детально проаналізовано отримані результати з точки зору зручності користування, доступності інтерфейсу та логічності структури сайту. Після завантаження головної сторінки (рис. 3.5) користувач одразу опиняється в інтуїтивно зрозумілому середовищі: у верхній частині розміщено логотип платформи, поле для пошуку товарів,

кошик для перегляду обраних позицій, зручний каталог продукції, а також кнопка входу до особистого облікового запису.

У центральній частині головної сторінки розташовано слоган «Ваш шопінг, ваше задоволення – все в одному місці», який не лише привертає увагу, а й відображає основну місію маркетплейсу – створити простір, де користувачі можуть здійснювати покупки швидко, зручно та із задоволенням. Завдяки цьому гаслу формується емоційний контакт із відвідувачем і підкреслюється орієнтація на комфорт споживача.

Нижче по сторінці розміщено розділ «Хіти продаж», у якому представлені найпопулярніші товари, що користуються підвищеним попитом серед покупців. Такий підхід дозволяє новим користувачам швидко ознайомитися з актуальними позиціями та полегшує процес вибору, водночас стимулюючи інтерес до продукції.

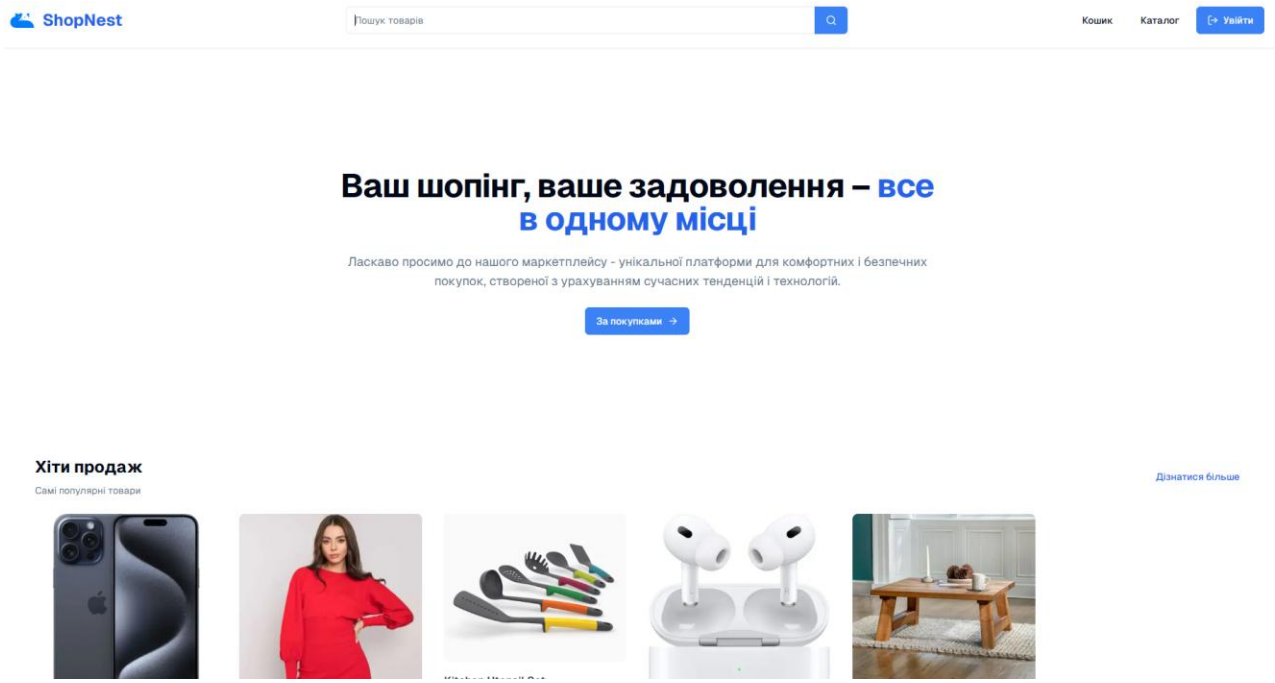

Рисунок 3.5 – Головна сторінка

У нижньому інформаційному блоці сторінки розташовано текст «hlumak © 2025 Усі права захищені» (рис. 3.6), що інформує користувача про авторські права на платформу та дату її опублікування. Назва «hlumak»

виконана як інтерактивне посилання, яке візуально виділено при наведенні курсору: шрифт змінює відтінок, з'являється підкреслення, що підказує про можливість кліку.

Після натискання на ім'я користувача автоматично перенаправляють до особистого телеграм-акаунту розробника. Такий механізм створює безпосередній канал комунікації, через який можна оперативно звернутися за технічною підтримкою, поставити питання щодо функціональних можливостей або залишити відгук про роботу платформи. Це особливо важливо для підтримання зворотного зв'язку та підвищення довіри між користувачем і автором проєкту.

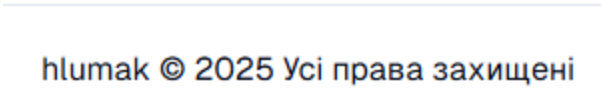

Рисунок 3.6 – Інформаційний блок

Після натискання на кнопку «Увійти» користувач автоматично перенаправляється на сторінку створення облікового запису (рис. 3.7).

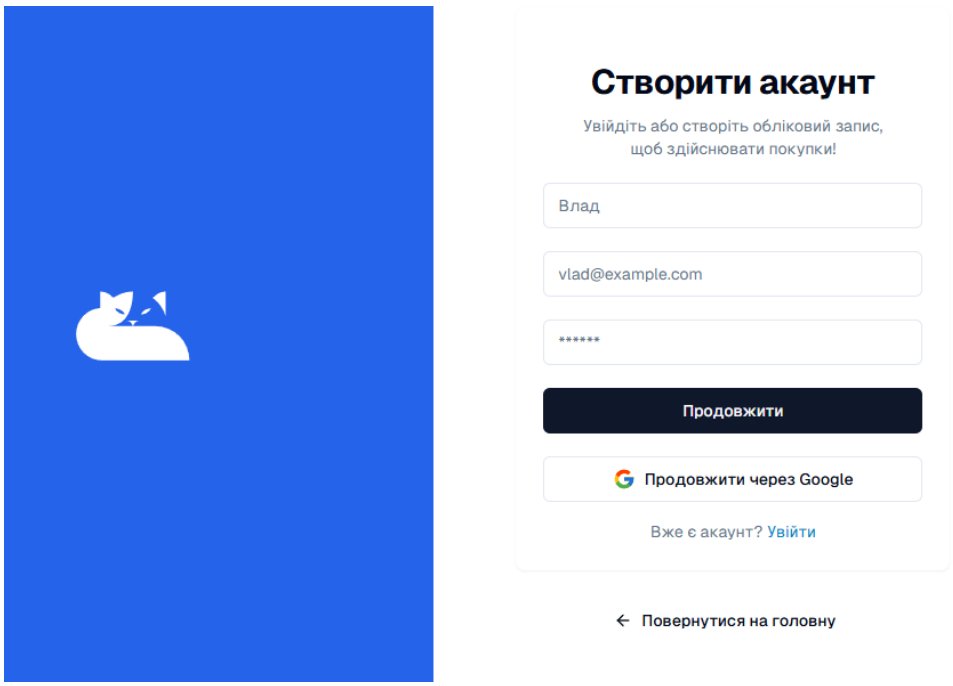

Рисунок 3.7 – Сторінка створення облікового запису

На цій сторінці необхідно ввести особисту інформацію: ім’я, адресу електронної пошти та пароль. Поля позначені зрозумілими підписами та містять підказки-плейсхолдери, що допомагають коректно заповнити форму. Особливу увагу слід приділити правильності введення цих даних. Зокрема, електронна адреса повинна бути дійсною, оскільки саме на неї можуть надсилатися підтвердження, сповіщення або інша важлива інформація. Пароль, у свою чергу, повинен містити щонайменше шість символів, щоб відповідати вимогам безпеки системи (рис. 3.8).

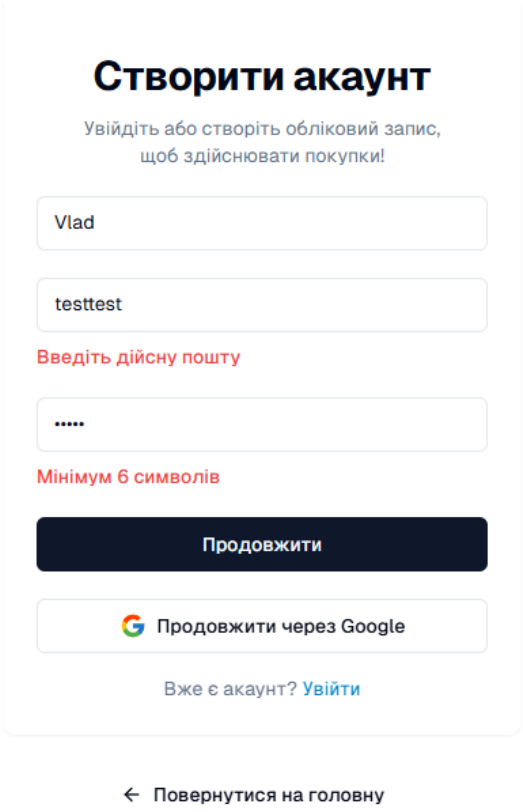

Рисунок 3.8 – Перевірка введених даних

У разі некоректного заповнення форми під відповідними полями з’являється контекстне повідомлення з поясненням помилки – наприклад, «Невірний формат електронної адреси» або «Пароль має містити не менше шести символів». Також передбачена можливість перегляду введеного пароля для уникнення помилок під час набору. Після успішного заповнення

всіх обов’язкових полів та натискання кнопки «Зареєструватися» користувач отримає спливаюче вікно з підтвердженням успішної реєстрації і автоматично перейде до персонального кабінету. Інтерфейс сторінки адаптивний: на мобільних пристроях форма займає всю ширину екрану, а кнопки та поля залишаються зручними для натискання пальцем.

У разі, якщо користувач вводить некоректні дані – наприклад, пошту без символу @ або занадто короткий пароль, то система миттєво повідомляє про помилку шляхом відображення текстового повідомлення під відповідним полем. Таким чином, форма забезпечує динамічну перевірку коректності введених даних, що підвищує зручність користування інтерфейсом.

Після успішної авторизації або створення облікового запису користувача автоматично перенаправляють до персонального кабінету (рис. 3.9). У верхній частині кабінету розташовано меню навігації, що містить такі вкладки:

– «Ваші замовлення» – таблиця з переліченням дат оплати, статусів замовлень та їхніх сум. Користувач може відсортувати записи за датою або сумою, натиснувши на відповідні заголовки стовпців;

– «Обране» – товари, позначені користувачем для подальшого перегляду або покупки;

– «Мої магазини» – перелік створених користувачем магазинів із можливістю швидкого переходу до керування кожним із них;

– «Налаштування профілю» – розділ для редагування особистих даних (ім’я, електронна адреса, пароль), а також керування сповіщеннями й безпекою облікового запису.

Під заголовком «Ваші замовлення» відображено таблицю, у якій кожен рядок відповідає окремому замовленню. Вона містить три основні колонки:

– «Дата оплати», «дата оформлення» та «оплати замовлення»;

– «Статус» – поточний етап обробки замовлення (наприклад, «В очікуванні», «Сплачено», «Відправлено»);

– «Сума» – загальна вартість замовлення в гривнях.

У разі відсутності замовлень користувач бачить повідомлення про порожній список, що робить інтерфейс зрозумілішим і запобігає відображенню некоректних даних.

Персональний кабінет виконаний у мінімалістичному стилі, без зайвих елементів, що сприяє швидкій орієнтації користувача. Інтерфейс адаптивний: на мобільних пристроях меню згортається в іконку, а таблиця замовлень перетворюється у перелік карток із базовою інформацією для зручного перегляду на невеликих екранах. Така структура забезпечує надійне та комфортне управління власним профілем і діяльністю в межах маркетплейсу.

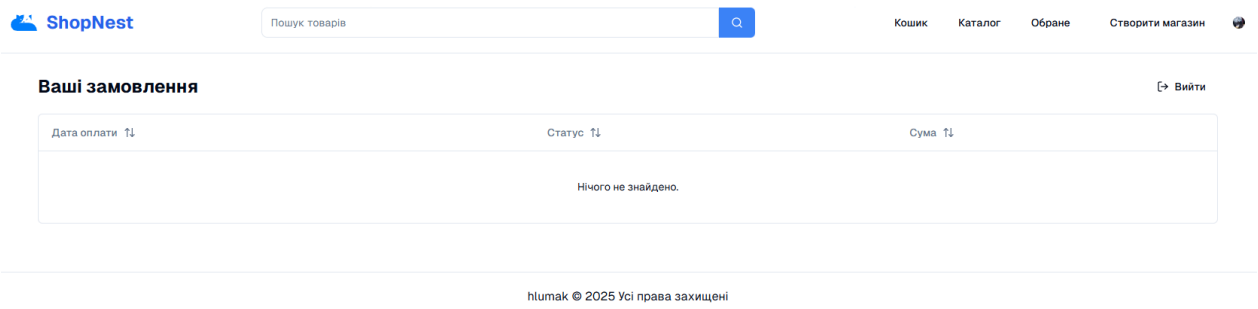

Рисунок 3.9 – Персональний кабінет користувача

Після входу до персонального кабінету користувача та створення магазину зі своєю назвою відкривається розділ статистики (рис. 3.10), який надає інформацію про основні показники діяльності магазину. На зображенні представлено початковий вигляд цього розділу. Інтерфейс відображає загальну інформацію про прибуток, кількість доданих товарів, кількість створених категорій, а також середній рейтинг магазину. У даному випадку усі значення дорівнюють нулю, оскільки магазин щойно створений і ще не має активних записів у відповідних розділах.

Під панеллю показників розміщено повідомлення, яке сповіщає користувача про відсутність даних для статистики. Під час тестування було встановлено, що система коректно реагує на початкову відсутність

активності, не допускаючи помилок чи некоректного відображення інформації. Усі елементи інтерфейсу завантажуються відповідно до очікувань, а повідомлення відображається в зрозумілій формі. Це свідчить про правильну реалізацію початкового стану статистичної панелі в межах вебзастосунку.

На боковій панелі розташовані основні елементи навігації, які забезпечують зручний доступ до ключових функціональних розділів системи. Зокрема, користувач може перейти до перегляду товарів, керування категоріями, налаштування кольорової гами, ознайомлення з відгуками, а також змінити параметри свого магазину. Така структура інтерфейсу дозволяє швидко орієнтуватися в системі та здійснювати необхідні дії без додаткових пошуків. Усі елементи розташовані логічно та зрозуміло для користувача, що позитивно впливає на загальну зручність користування платформою.

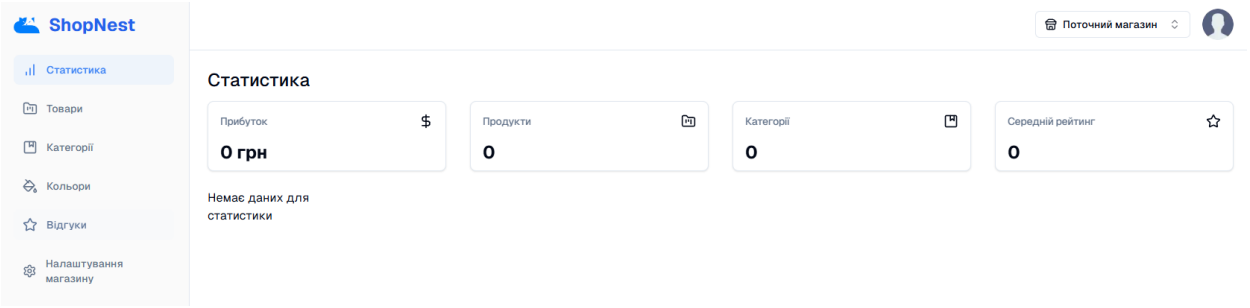

Рисунок 3.10 – Розділ статистики

На боковій панелі розташовані основні елементи навігації, які забезпечують зручний доступ до ключових функціональних розділів системи. Завдяки логічній структурі та продуманому розміщенню, користувач має змогу оперативно переходити до різних розділів, не витрачаючи час на пошук потрібної інформації. Зокрема, на панелі передбачено швидкий доступ до перегляду товарів, керування категоріями, налаштування кольорової гами, перегляду та модерації відгуків, а також до зміни параметрів магазину, включаючи контактну інформацію, логотип або інші візуальні елементи.

Така структура інтерфейсу дозволяє швидко орієнтуватися в системі та здійснювати необхідні дії без додаткових пошуків або складних переходів між сторінками. Усі елементи розташовані інтуїтивно зрозуміло, що значно підвищує зручність користування платформою та зменшує поріг входу для нових користувачів.

Процес створення товару для вашого магазину є максимально зручним, легким та інтуїтивно зрозумілим. Усі необхідні дії можна виконати в одному вікні без потреби перемикатися між різними вкладками чи модулями. Це значно пришвидшує процес додавання товарів та підвищує ефективність управління контентом магазину. Під час додавання нового товару немає потреби попередньо створювати категорії чи кольори – усі ці дії можна виконати безпосередньо в процесі заповнення форми. Наприклад, при необхідності можна одразу створити нову категорію, задати унікальний колір або визначити інші характеристики, такі як розміри, матеріали чи додаткові опції. Така гнучкість є особливо корисною для тих користувачів, які щойно починають працювати з платформою або активно розширюють асортимент продукції. В результаті, час на додавання нового товару скорочується, а можливість налаштовувати кожну позицію індивідуально відкриває широкі можливості для персоналізації вітрини магазину.

Процес пошуку на сторінці реалізований коректно та надійно, забезпечуючи високу зручність для користувача (рис. 3.11). Інтерфейс пошуку дозволяє миттєво отримати результати навіть за умов неточного введення запиту. Якщо введена в пошуковий рядок назва товару відсутня у базі даних, система негайно виводить інформативне повідомлення про відсутність результатів, що запобігає непорозумінням. У випадку, коли користувач вводить хоча б одне слово або фрагмент назви, який частково збігається з існуючими товарами, система автоматично формує список результатів, які найбільше відповідають пошуковому запиту. Це дає змогу швидко знаходити необхідні товари навіть при використанні неточних або неповних назв, що значно покращує ефективність пошуку.

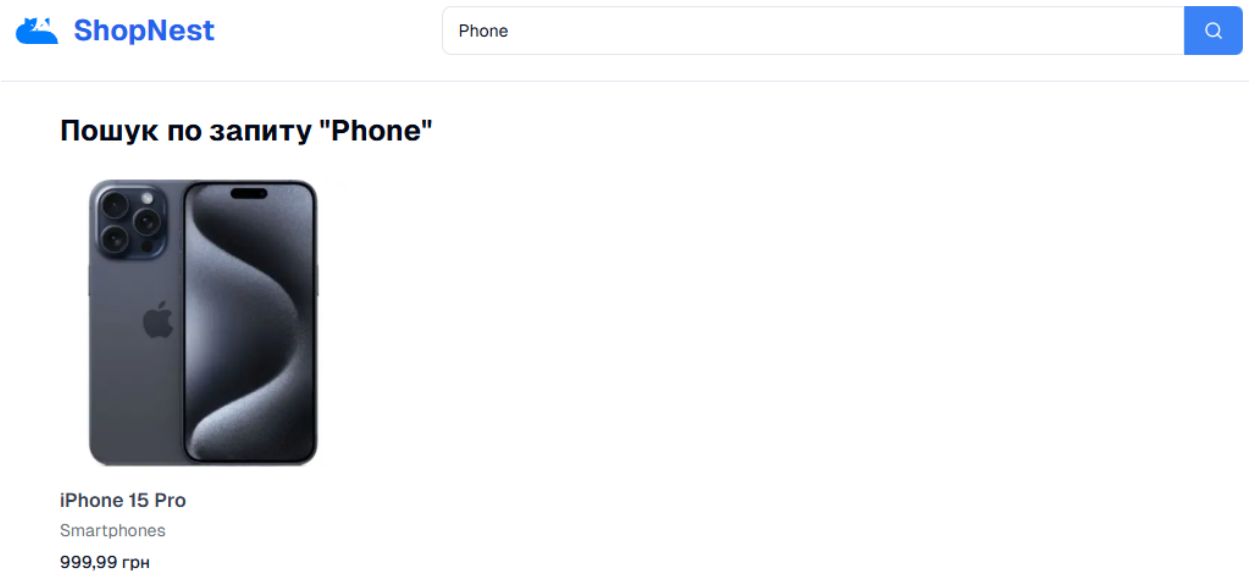

Рисунок 3.11 – Процес пошуку

Процес додавання товарів до кошику та оплата проходять без жодних труднощів. Користувач може легко і швидко додавати обрані товари як до кошику, так і до списку «Обране» для подальшого перегляду або покупки. Оплата здійснюється без помилок і затримок, забезпечуючи комфортний та безпечний досвід покупки. Наразі система оплати працює у тестовому режимі, що дає змогу перевірити всі функції перед запуском у повноцінну експлуатацію.

Окрім загального функціонального тестування, було також проведено кросбраузерне тестування для перевірки коректності роботи вебзастосунку в різних середовищах перегляду. Застосунок тестувався у таких популярних браузерах, як Google Chrome, Microsoft Edge та Opera. Метою цього етапу було переконатися, що інтерфейс користувача однаково правильно відображається, елементи дизайну не зміщуються, а функціональні компоненти працюють стабільно та без збоїв у кожному з обраних браузерів.

У результаті тестування встановлено, що вебзастосунок демонструє однакову поведінку у всіх перевірених браузерах. Візуальні елементи, зокрема кнопки, форми, меню, іконки та панелі, завантажуються без помилок, не змінюють свого розташування і не порушують загальної

структури сторінки. Анімації, динамічні елементи, а також інтерактивні функції, такі як пошук, додавання товарів до кошика або авторизація, виконуються коректно і без затримок.

Також було перевірено адаптивність дизайну, зокрема правильність відображення сайту на різних розширеннях екрана у вказаних браузерах. У всіх випадках сторінки відображались згідно з очікуваннями, не втрачаючи ключових елементів або функціональності. Це свідчить про належну реалізацію принципів кросбраузерної сумісності та дає змогу користувачам працювати із платформою комфортно незалежно від обраного браузера.

Отже, результати проведеного тестування вебзастосунку дали змогу здійснити всебічну оцінку його функціональності, зручності користування та коректності роботи основних елементів інтерфейсу. У процесі тестування було перевірено ключові сценарії взаємодії користувача з платформою – від реєстрації та входу в обліковий запис до створення магазину, додавання товарів, їх пошуку, додавання до кошика, а також здійснення тестової оплати. Усі ці етапи були реалізовані з дотриманням принципів логічної послідовності, інтуїтивної зрозумілості та високої стабільності роботи.

Інтерфейс вебзастосунку відзначається структурованістю та логічною організацією: усі елементи розміщені у звичних для користувача місцях, що забезпечує швидке засвоєння функціоналу та комфортну роботу з платформою навіть для нових користувачів. Система успішно обробляє як стандартні дії, так і помилкові введення, своєчасно надаючи інформативні повідомлення, які допомагають усунути неточності без необхідності додаткової допомоги. Це свідчить про наявність надійних механізмів валідації введених даних та високий рівень продуманості інтерфейсу.

Окрему увагу було приділено перевірці кросбраузерної сумісності. Вебзастосунок успішно пройшов тестування у найпоширеніших браузерах, зокрема Google Chrome, Microsoft Edge та Opera. Усі функції працювали стабільно, без помилок чи збоїв, а елементи дизайну залишались коректно відображеними незалежно від середовища перегляду. Це підтверджує

універсальність інтерфейсу та його готовність до повсякденного використання широким колом користувачів.

Функціональні модулі, такі як персональний кабінет, розділ статистики, панель керування магазином, а також можливість створення товарів із супутніми характеристиками без необхідності переходу в інші розділи, працюють належним чином і відповідають очікуванням. Також було підтверджено правильність роботи пошукової системи, яка забезпечує виведення релевантних результатів навіть при частковому введенні запиту, що значно полегшує навігацію по товарах.

Загалом, можна зробити висновок, що вебзастосунок створено відповідно до актуальних вимог щодо функціональності, зручності та доступності. Його архітектура, логіка взаємодії та технічна реалізація демонструють високу якість, що дозволяє впевнено рекомендувати платформу до подальшого впровадження, масштабування та повноцінної експлуатації в реальних умовах.

3.4 Перспективи подальшої розробки маркетплейсу

У процесі створення маркетплейсу ShopNest було закладено надійну основу для подальшого розвитку проєкту. Одним із ключових напрямків майбутньої роботи є розширення функціоналу платформи. Зокрема, доцільно впровадити модуль аналітики для продавців, що дозволить їм відстежувати статистику продажів, аналізувати популярність товарів та динаміку замовлень. Окрім цього, важливим кроком стане розробка системи персоналізованих рекомендацій для покупців, яка базуватиметься на їхній поведінці та історії покупок. Для залучення ширшої аудиторії варто реалізувати підтримку багатомовності інтерфейсу.

Подальший розвиток проєкту передбачає інтеграцію зі сторонніми сервісами, зокрема підключення додаткових сучасних платіжних систем,

таких як Stripe, PayPal або MonoPay, що забезпечить зручність та безпеку фінансових операцій. Також актуальним є підключення сервісів доставки, наприклад, Нової Пошти чи Укрпошти, для автоматизації логістичних процесів.

Особливу увагу слід приділити підвищенню рівня безпеки та масштабованості системи. Впровадження двофакторної аутентифікації значно підвищить захищеність облікових записів користувачів. Оптимізація продуктивності та підготовка до роботи з великим навантаженням дозволить платформі ефективно функціонувати навіть за умов зростання кількості користувачів. Перехід до мікросервісної архітектури забезпечить гнучкість та спростить подальше масштабування проєкту.

Ще одним перспективним напрямком є розробка мобільного застосунку, що підвищить доступність сервісу для користувачів та дозволить їм здійснювати покупки з будь-яких пристроїв. Окрім цього, доцільно вдосконалити адміністративну панель, розширивши її можливості для модерації контенту, управління користувачами та перегляду аналітики.

Покращення користувацького досвіду також залишається пріоритетом. Проведення A/B тестування допоможе оптимізувати інтерфейс, а впровадження темної теми та адаптивного дизайну зробить платформу зручнішою для користувачів на різних пристроях. Реалізація зазначених напрямків дозволить підвищити конкурентоспроможність маркетплейсу ShopNest, розширити його функціональні можливості та забезпечити якісний користувацький досвід.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований вебмаркетплейс для електронної комерції з можливістю створення користувацьких магазинів, який дозволяє користувачам здійснювати покупки, а продавцям – створювати власні онлайн-магазини, розміщувати товари, керувати контентом та аналітикою.

У ході виконання роботи проведено аналіз сучасних рішень у сфері e-commerce, визначено основні вимоги до функціоналу та архітектури маркетплейсу. На основі цього було реалізовано клієнтську та серверну частини застосунку із сучасними підходами, зокрема SPA-архітектурою, REST API, та зручним інтерфейсом. Були використані фреймворки NextJS та NestJS, а також база даних PostgreSQL.

Платформа підтримує базову реєстрацію користувачів, створення магазинів, додавання товарів, їхнє редагування, перегляд статистики та обробку замовлень. Додаткову увагу приділено зручності користувацького досвіду, адаптивному дизайну та системі пошуку.

За результатами тестування підтверджено стабільну роботу основних функцій маркетплейсу на різних пристроях. Запропоновано напрями подальшого розвитку, зокрема: впровадження мобільного застосунку, вдосконалення адміністративної панелі, покращення захисту персональних даних і впровадження додаткових інструментів аналітики.

Реалізований застосунок має практичну цінність і може бути основою для впровадження у реальних бізнес-процесах малого та середнього підприємництва.

Результати роботи апробовано у вигляді тез доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [30].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Петров, Е. Г., Новожилова, М. В., Гребеннік, І. В., & Соколова, Н. А. (2003). *Методи та засоби прийняття рішень у соціально-економічних та технічних системах*. Херсон: Олді–плюс.

2. Гайдар М. І. Дослідження застосування методів аналізу даних в системах електронної комерції. Пояснювальна записка до атестаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 122 – Комп'ютерні науки. М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2021. – 75 с.

3. Петренко, М. А. Управління безпекою діяльності e-commerce підприємств. Пояснювальна записка до атестаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 073 – Менеджмент. М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2022. – 90 с.

4. Гребеннік, І. В., Коваленко, А. І., Міщеряков, Ю. В., Решетнік, В. М., & Тітов, С. В. (2017). *Системне програмування*. Харків: ХНУРЕ.

5. Тихонов І. О. Розробка системи електронної комерції послуг каршерингу. Радіоелектроніка та молодь у XXI столітті: матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. – Харків : ХНУРЕ, 2024. – Т. 6 – С. 738-740.

6. Міжнародна науково-практична конференція «Математичне моделювання процесів в економіці та управлінні проектами і програмами (ММП-2019)», Коблево, 9-13 вересня 2019 р. Праці – Харків: ХНУРЕ, 2019. – 151 с.

7. Просування на маркетплейсах: який майданчик обрати та чому?. Udigital. URL: https://udigital.com.ua/prosuvannya-na-marketplejsah-yakij-majdanchik-obrati-ta-chomu (дата звернення 20.04.2025).

8. Найкращі торгові майданчики для продажу в інтернеті: приклади в Україні та світі | SendPulse UA. Блог про email та інтернет-маркетінг. URL: https://sendpulse.ua/blog/best-shopping-areas (дата звернення 20.04.2025).

9. Ukrinform. Rozetka + Prom.ua: як і проти кого “дружитимуть” гранди української е-комерції. Укрінформ - актуальні новини України та світу. URL: https://www.ukrinform.ua/rubric-economy/2522026-rozetka-promua-ak-i-proti-kogo-druzitimut-grandi-ukrainskoi-ekomercii.html (дата звернення 20.04.2025).

10. В чому секрет продажів на Розетка? - KeyCRM Blog. KeyCRM Blog. URL: https://blog.keycrm.app/uk/v-chomu-sekret-prodazhiv-na-rozetka (дата звернення 20.04.2025).

11. Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley.

12. Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.

13. Atkinson, B., & Castro, R. (2021). *Software Architecture Patterns for Serverless Systems*. O'Reilly Media.

14. Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California, Irvine).

15. Masse, M. (2011). *REST API design rulebook*. O'Reilly Media.

16. DOU. (2025, 10 лютого). Рейтинг мов програмування 2025. TypeScript і Python — найпопулярніші, частки C# та Java зменшуються. DOU. URL: https://dou.ua/lenta/articles/language-rating-2025 (дата звернення 25.04.2025).

17. Fastify. Fast and low overhead web framework, for Node.js | Fastify. URL: https://fastify.dev/benchmarks (дата звернення 25.04.2025).

18. Reenskaug, T. (1979). *Models-views-controllers* (Xerox PARC Technical Note).

19. Wieruch, R. (2019). *The road to React: Your journey to master React.js in JavaScript* (Latest ed.). Independently published.

20. *DB-Engines Ranking*. DB-Engines. URL: https://db-engines.com/en/ranking (дата звернення 26.04.2025).

21. *Documentation | NestJS - A progressive Node.js framework*. (б. д.). Documentation | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com (дата звернення 15.05.2025).

22. *Prisma Documentation*. Prisma | Instant Postgres plus an ORM for simpler db workflows. URL: https://www.prisma.io/docs (дата звернення 15.05.2025).

23. *PostgreSQL: Documentation*. PostgreSQL: The world's most advanced open source database. URL: https://www.postgresql.org/docs (дата звернення 15.05.2025).

24. *Introduction | Fastify*. Fast and low overhead web framework, for Node.js | Fastify. URL: https://fastify.dev/docs/latest (дата звернення: 16.05.2025).

25. Jones, M. B., Bradley, J., & Sakimura, N. *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force (IETF). URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 16.05.2025).

26. De Laet, J. (2022). *Mastering Next.js: Build server-side rendering apps with Next.js 13 and App Router*. Leanpub.

27. Redux Toolkit Team. *Redux Toolkit: Efficient Redux development*. URL: https://redux-toolkit.js.org (дата звернення: 16.05.2025).

28. TanStack. *React Query Documentation*. URL: https://tanstack.com/query/latest (дата звернення: 17.05.2025).

29. Zod Authors. *Zod: TypeScript-first schema validation with static type inference*. URL: https://zod.dev (дата звернення: 17.05.2025).

30. Глумаков В.О. (2025) Розробка веб-маркетплейсу для електронної комерції з можливістю створення користувацьких магазинів. *Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–18 квітня 2025 р.).* Харків: ХНУРЕ, 2025. Т. 6. С. 393-394.