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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: **«Проєктування клієнтської частини комерційного вебсайту інтернет-магазину цифрової техніки з використанням React та TypeScript»**

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_2021
Костов Михайло Михайлович
Керівник: Поночовний Юрій
Леонідович
Рецензент: Муравльов Володимир
В'ячеславович

Полтава – 2025 року

ВСТУП

Актуальність. З розвитком цифрових технологій комерційні інтернет-магазини набули особливої популярності. Онлайн-комерція стала невіддільною частиною життя споживачів, надаючи зручні умови вибору товарів, економію часу та широкий спектр інформаційного забезпечення покупки. Особливу актуальність мають інтернет-магазини цифрової техніки, де важливо надати покупцеві максимально комфортні умови взаємодії з інтерфейсом, що безпосередньо впливає на вибір та прийняття рішення про покупку. Сучасні технології розробки фронтенду, такі як React у поєднанні з TypeScript, дають змогу створювати продуктивні, динамічні та надійні користувацькі інтерфейси, які відповідають високим стандартам якості та безпеки. Саме тому дослідження та проєктування клієнтської частини вебсайтів електронної комерції на базі цих технологій є надзвичайно актуальним завданням сучасної веброзробки.

Аналіз стану розробки проблеми. Нині існує велика кількість наукових праць, статей та практичних досліджень, що стосуються створення вебсайтів за допомогою React та TypeScript. Важливий внесок зроблено у сфері використання компонентного підходу та методології декларативного програмування, зокрема в роботах зарубіжних і вітчизняних дослідників та розробників.

Метою кваліфікаційної роботи є проєктування та реалізація клієнтської частини комерційного вебсайту інтернет-магазину цифрової техніки із застосуванням сучасних технологій React і TypeScript.

Для досягнення зазначеної мети були поставлені наступні завдання:

1) провести аналіз сучасних технологій фронтенд-розробки та обґрунтувати вибір React і TypeScript;

2) вивчити основні принципи розробки інтерфейсів інтернет-магазинів;

3) спроєктувати архітектуру та структуру клієнтської частини вебсайту;

4) реалізувати базові та додаткові функціональні можливості користувацького інтерфейсу;

5) провести тестування створеного вебсайту.

Об'єкт дослідження – процес проєктування та розробки клієнтської частини вебсайтів інтернет-магазинів.

Предмет дослідження – методики та технології проєктування клієнтської частини комерційного вебсайту цифрової техніки з використанням React та TypeScript.

Методи наукових досліджень. Для вирішення поставлених завдань у кваліфікаційної роботі були використані такі методи:

1. Аналіз та синтез – для вивчення сучасного стану технологій та обґрунтування їх вибору.

2. Метод моделювання – для проєктування структури вебзастосунку.

3. Експериментальний метод – для практичної реалізації вебсайту.

4. Тестування – для оцінки працездатності та ефективності реалізованого вебзастосунку.

Інформаційна база дослідження містить спеціалізовану літературу з питань веброзробки, документацію до React та TypeScript, інформацію з офіційних сайтів розробників технологій, наукові статті, навчальні матеріали, а також приклади реалізації подібних вебзастосунків.

Практична значущість роботи полягає в тому, що розроблено повністю функціональну клієнтську частину вебсайту інтернет-магазину цифрової техніки, що забезпечує зручність та інтуїтивність користування, швидкодію та легкість подальшого масштабування.

Результати роботи апробовані в рамках наукової конференції здобувачів вищої освіти за результатами науково-дослідної роботи у 2024-2025 рр.

Структура та обсяг кваліфікаційної роботи. Робота містить перелік умовних позначень, вступ, три розділи основної частини, висновки, список використаних джерел. Загальний обсяг роботи становить 47 сторінок, містить 9 рисунків, 4 таблиці, список використаних джерел нараховує 30 найменувань.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ТЕХНОЛОГІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБСАЙТУ

1.1 Огляд технологій для розробки клієнтської частини вебсайту

У сучасній веброзробці ключову роль відіграють інструменти, які дозволяють створювати високопродуктивні та зручні вебзастосунки. Однією з найпопулярніших бібліотек для створення інтерфейсів є React.

React [1] – це JavaScript-бібліотека з відкритим вихідним кодом, яка була розроблена компанією Facebook і призначена для створення користувацьких інтерфейсів. Основою React є компонентний підхід, що дозволяє розбивати інтерфейс на незалежні, багаторазові частини (компоненти). Кожен компонент описує частину інтерфейсу, має власний стан (state) та властивості (props), які використовуються для передачі даних між компонентами.

JSX – спеціалізований синтаксис, що поєднує JavaScript з HTML-подібною розміткою, значно спрощує написання коду та підвищує його читабельність. Для керування станом компонентів в React часто використовують вбудовані хуки (useState, useEffect), а також додаткові бібліотеки, такі як Redux, які дозволяють централізовано управляти станом складних застосунків.

Для поліпшення надійності та простоти супроводу коду в проєктах на базі React все частіше використовується TypeScript. TypeScript [2] – це надбудова над JavaScript, яка додає статичну типізацію, що дозволяє виявляти потенційні помилки вже на етапі написання коду. Основні переваги використання TypeScript у проєктах React [3]:

- підвищення якості коду завдяки чіткій типізації;
- зменшення кількості помилок на етапі виконання програми;
- покращення автодоповнення і підказок у редакторах коду;
- легша підтримка та розширення існуючих кодових баз.

Інтеграція TypeScript з React здійснюється через налаштування спеціального шаблону проєкту або вручну через конфігурацію tsconfig.json, де вказуються параметри компіляції та налаштування поведінки TypeScript. Поширеними інструментами для цього є Create React App [4] з шаблоном для TypeScript, що дозволяє швидко стартувати проєкти з мінімальними конфігураціями.

Крім React та TypeScript, для повноцінної розробки клієнтської частини вебсайту використовується низка інших базових технологій та інструментів (рис. 1.1):

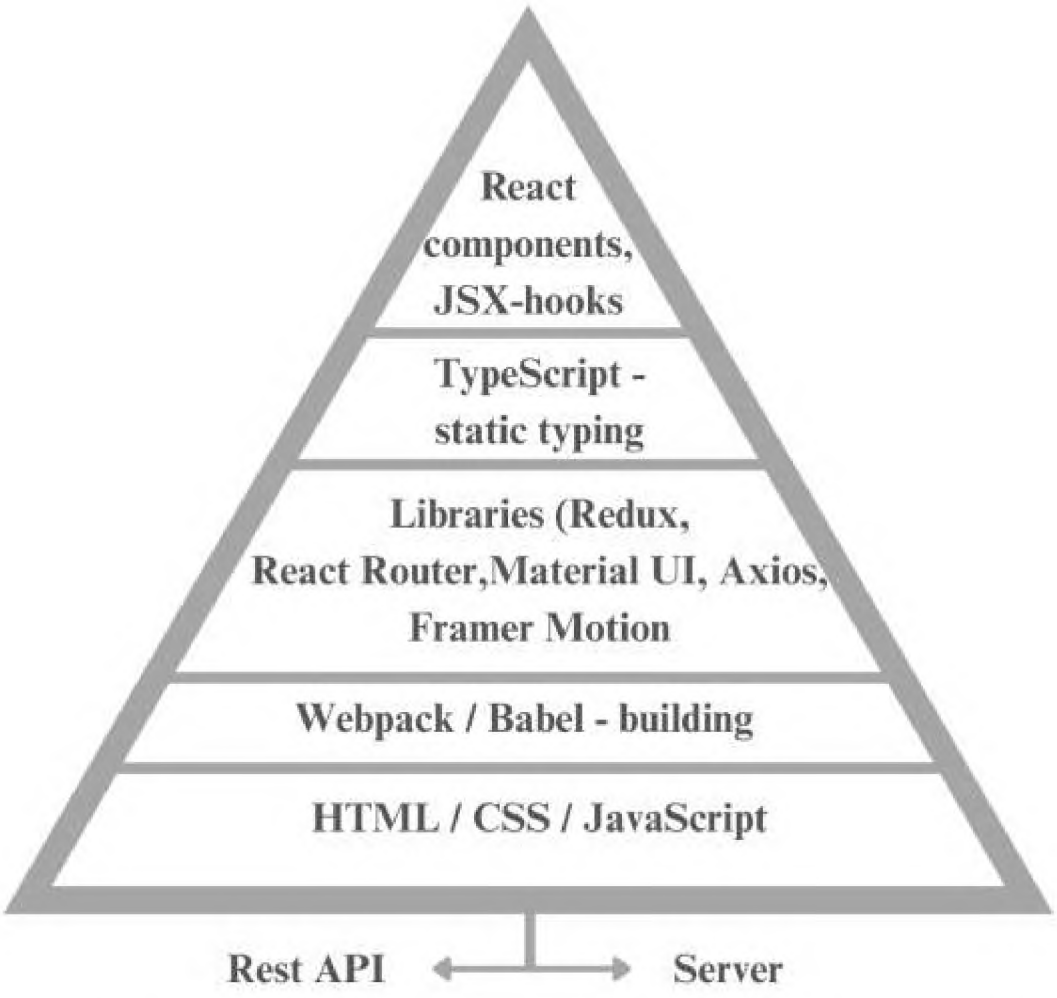

Рисунок 1.1 – Схема технологічного стека клієнтської частини (React + TypeScript)

– HTML (HyperText Markup Language) є фундаментальною технологією, яка використовується для побудови структури вебсторінок.

– CSS (Cascading Style Sheets) відповідає за стилізацію вебсайтів, забезпечує візуальне оформлення інтерфейсів.

– JavaScript – основна мова програмування, що використовується для додавання інтерактивності сторінкам.

– Webpack – потужний інструмент для збирання модулів, дозволяє об'єднувати, оптимізувати та управляти залежностями у фронтенд-застосунках.

– Babel – транспайлер, який перетворює сучасний код JavaScript та JSX у код, сумісний із більш старими версіями браузерів.

– Redux – бібліотека для управління станом у React-застосунках, яка дозволяє централізувати і спростити управління даними в великих проектах.

Взаємодія клієнтської частини з сервером зазвичай здійснюється за допомогою RESTful API REST (Representational State Transfer) [5] є архітектурним стилем, що визначає набір правил створення вебсервісів. Вебсервіси, побудовані за REST-архітектурою, характеризуються чітко визначеними методами HTTP-запитів (GET, POST, PUT, DELETE), які виконують відповідні операції з ресурсами. Робота з RESTful API в React-застосунках часто здійснюється за допомогою бібліотеки Axios або вбудованого API fetch(), що забезпечує асинхронне завантаження даних і їхню динамічну обробку.

Окрім базових технологій, важливу роль у підвищенні ефективності розробки і покращенні UX/UI грають додаткові бібліотеки. До таких належать:

– React Router [6] – бібліотека для організації маршрутизації у застосунках React, що дозволяє створювати багатосторінкові інтерфейси.

– Material UI, Ant Design, Bootstrap – популярні бібліотеки компонентів для створення зручних та естетично привабливих інтерфейсів з мінімальними витратами часу.

– Framer Motion – інструмент для додавання анімації в React-компоненти, що значно поліпшує візуальний досвід користувачів.

Отже, вибір технологій React і TypeScript, а також додаткових інструментів та бібліотек є логічним та виправданим для створення якісної клієнтської частини вебсайту інтернет-магазину цифрової техніки. Поєднання цих технологій дозволяє досягати високих стандартів якості коду, ефективності розробки та зручності використання кінцевими користувачами.

1.2 Аналіз конкурентних рішень на ринку інтернет-магазинів цифрової техніки

Сучасний ринок інтернет-магазинів цифрової техніки є одним з найбільш динамічних сегментів електронної комерції, характеризується високою конкуренцією та активним використанням інноваційних технологій. Для досягнення успіху новий інтернет-магазин повинен орієнтуватися на кращі приклади, вивчати недоліки конкурентів та враховувати сучасні тенденції у розробці вебсайтів.

Огляд основних конкурентів у сфері продажу цифрової техніки через Інтернет дозволяє виділити такі популярні платформи, як Rozetka, Comfy, Citrus, MOYO та Eldorado. Ці компанії володіють значною часткою українського ринку і вже тривалий час демонструють стабільне зростання завдяки активному використанню сучасних вебтехнологій, ефективній маркетинговій стратегії та високій якості обслуговування клієнтів.

Провівши порівняльний аналіз технологій та функціональних можливостей цих конкурентних вебсайтів, можна зазначити, що більшість із них використовує React або аналогічні JavaScript-фреймворки для створення динамічних та адаптивних користувацьких інтерфейсів. Rozetka, як один з лідерів ринку, активно застосовує сучасні підходи до розробки, пропонуючи зручну навігацію, швидкий пошук товарів, інтегровану систему відгуків, детальну інформацію про продукти та оптимізовану мобільну версію сайту. На рис. 1.2 наведено знімок екрану головної сторінки Rozetka.

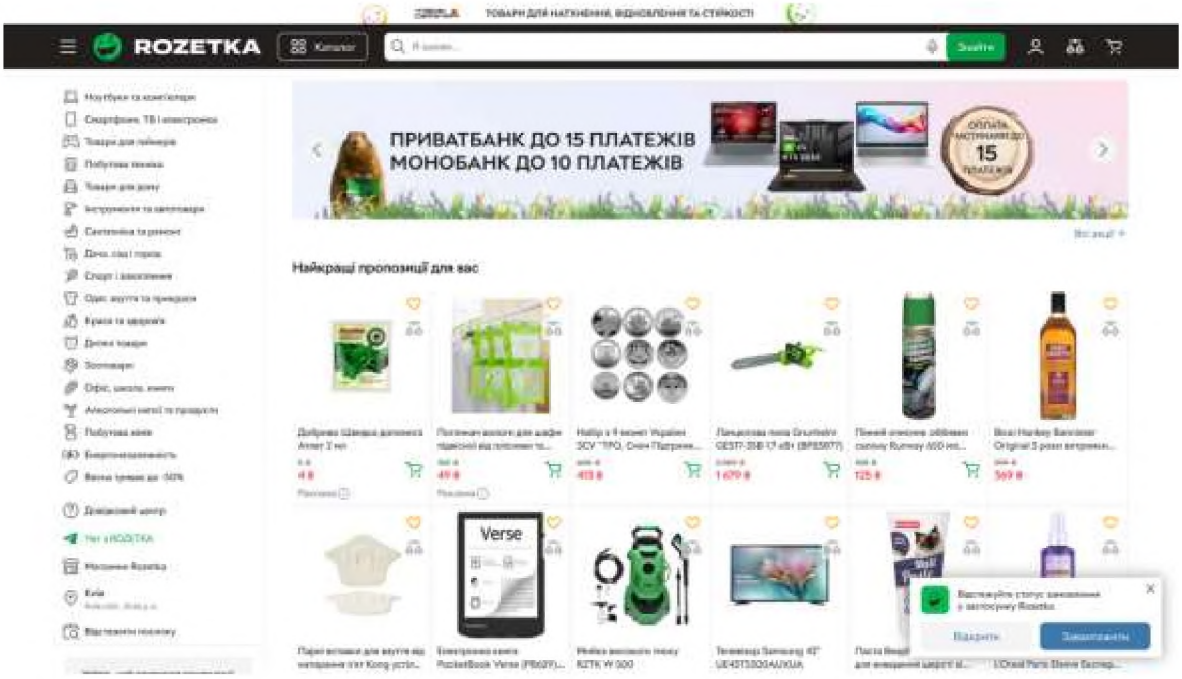

Рисунок 1.2 – Скріншот екрану вебсторінки Rozetka

Citrus та Comfy також вирізняються високою швидкодією вебсайтів, інтерактивними елементами та розширеними можливостями для персоналізації покупок.

Функціональні можливості конкурентних вебсайтів охоплюють широкий спектр: від простих каталогів з товарами до складних систем персональних рекомендацій, розвинених кошиків покупок та інтеграції різноманітних платіжних систем. Зокрема, популярними є можливості порівняння товарів, швидкого оформлення замовлення без реєстрації, інтерактивні консультації у режимі онлайн-чату та інтеграція соціальних мереж для швидкого входу і спілкування. Також активно використовується SEO-оптимізація, яка забезпечує високі позиції у пошукових системах.

Визначення сильних та слабких сторін конкурентних рішень є необхідним для створення успішного власного інтернет-магазину. До сильних сторін конкурентів можна віднести:

– високий рівень технологічності сайтів, використання новітніх технологій фронтенд-розробки;

– широкий функціонал, який забезпечує зручність та швидкість здійснення покупок;

– інтеграція із сучасними платіжними сервісами та службами доставки;

– активна комунікація з клієнтами через соціальні мережі та чати;

– використання персоналізації, рекомендаційних систем для підвищення середнього чеку та задоволеності клієнтів.

Однак існують і слабкі сторони, які можуть бути використані для отримання конкурентної переваги. Серед основних недоліків можна виділити:

– надмірна завантаженість інтерфейсу, що іноді ускладнює навігацію;

– недостатньо оптимізована мобільна версія на деяких платформах, що може негативно впливати на користувацький досвід мобільних покупців;

– не завжди ефективне вирішення питань технічної підтримки та рекламаційних звернень;

– обмежена гнучкість у кастомізації продуктів, коли користувачі мають специфічні вимоги до товарів.

Таким чином, аналіз конкурентних рішень дозволяє сформулювати ключові напрями розвитку власного вебсайту, що враховують досвід лідерів галузі та дозволяють уникнути їх помилок. Рекомендується особливу увагу приділити спрощенню користувацького інтерфейсу, ефективному використанню мобільних технологій, а також впровадженню ефективної системи підтримки клієнтів, що значно покращить загальну конкурентоспроможність нового інтернет-магазину.

1.3 Формулювання задачі дослідження

Для успішної реалізації клієнтської частини комерційного вебсайту інтернет-магазину цифрової техніки необхідно чітко визначити основні вимоги до функціональності, поставити завдання з розробки інтерфейсу користувача та окреслити вимоги до ефективності і безпеки. Визначення цих параметрів є

критично важливим етапом, що значно впливає на якість кінцевого продукту та загальну задоволеність користувачів.

Основними вимогами до функціональності клієнтської частини сайту є забезпечення інтуїтивно зрозумілої навігації, яка дає змогу легко орієнтуватися в асортименті товарів, пошук необхідних позицій, а також детальна інформація про товари з можливістю зміни їх характеристик (наприклад, вибір кольору чи обсягу пам'яті). Важливим елементом функціональності є також зручна система управління кошиком покупок та оформлення замовлень, яка мінімізує кількість дій користувача і робить процес купівлі максимально простим та швидким. Додатковими важливими функціями є можливість реєстрації користувачів для кращого управління замовленнями та інтеграція зручної системи відгуків і оцінок для товарів.

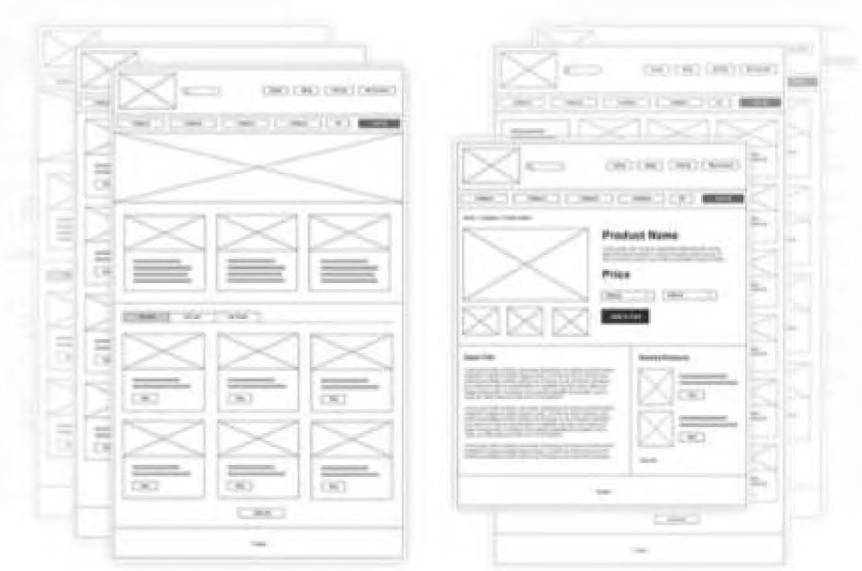

Рисунок 1.3 – Приклад wireframe-макету каталогу та сторінки товару інтернет-магазину

Завдання щодо розробки інтерфейсу користувача мають бути орієнтовані на створення максимально зручного і зрозумілого інтерфейсу, який відповідає сучасним стандартам UX/UI-дизайну (рис. 1.3). Wireframe-ескіз [7] – це спрощений схематичний макет сторінки без використання кольорів і стилів, що демонструє розташування основних блоків та елементів інтерфейсу.

Першочерговим є забезпечення адаптивності дизайну, що гарантує однаково якісний досвід користування сайтом на різних пристроях. Особлива увага приділяється оптимізації швидкості завантаження сторінок [8], оскільки це безпосередньо впливає на задоволеність користувачів і їхню готовність завершити покупку. Інтеграція інтерактивних елементів, які підвищують залучення користувачів, а також прозора структура каталогу та прості форми оформлення замовлень залишаються критично важливими аспектами цифрового інтерфейсу.

Вимоги до ефективності та безпеки вебсайту необхідні для забезпечення стабільної роботи, захисту інформації та довіри користувачів. В контексті ефективності особлива увага приділяється швидкодії вебзастосунку, яка повинна забезпечити час завантаження сторінок до 2-3 секунд, а також масштабованості системи для роботи з великими обсягами даних та користувачів. Надійність роботи сайту має підтримуватися через регулярний моніторинг і оновлення програмного забезпечення.

Безпека вебсайту передбачає захист персональних даних користувачів, реалізацію захищених з'єднань через HTTPS, використання безпечних методів зберігання конфіденційних даних (наприклад, хешування паролів), а також захист від потенційних кіберзагроз (SQL-ін'єкції, XSS, CSRF). Регулярне оновлення програмного забезпечення є важливою частиною забезпечення належного рівня безпеки та відповідності міжнародним стандартам.

Проведений аналіз теоретичних основ та технологічних рішень для розробки клієнтської частини вебсайту засвідчив, що використання бібліотеки React у поєднанні з TypeScript є оптимальним вибором для створення високопродуктивних, масштабованих та зручних вебзастосунків. Компонентний підхід React, доповнений статичною типізацією TypeScript, забезпечує високу якість коду, спрощує його підтримку та мінімізує помилки на етапі розробки. Додаткові інструменти, такі як Redux, Webpack, Babel, а також бібліотеки для маршрутизації (React Router) та стилізації (Material UI, Bootstrap), дозволяють створювати інтерактивні та естетично привабливі інтерфейси. Аналіз

конкурентних рішень на ринку інтернет-магазинів цифрової техніки виявив сильні сторони лідерів галузі, зокрема використання сучасних фронтенд-технологій та розвиненого функціоналу, а також слабкі сторони, такі як перевантаженість інтерфейсів чи недостатня оптимізація мобільних версій, що відкриває можливості для конкурентної переваги. Сформульовані вимоги до функціональності, UX/UI-дизайну, ефективності та безпеки клієнтської частини вебсайту інтернет-магазину цифрової техніки створюють чітке підґрунтя для подальшої розробки, орієнтованої на зручність користувачів, швидкодію та захист даних.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБСАЙТУ. АНАЛІЗ ВИМОГ ТА РОЗРОБКА АРХІТЕКТУРИ

2.1 Аналіз потреб користувачів та розробка user-case сценаріїв

Проєктування клієнтської частини комерційного вебсайту інтернет-магазину цифрової техніки розпочинається з ретельного аналізу потреб його потенційних користувачів. Для успішного функціонування вебсайту необхідно зрозуміти, які категорії користувачів будуть взаємодіяти з ресурсом, яким чином вони будуть використовувати сайт, а також визначити ключові функції, необхідні для забезпечення їхніх потреб та очікувань. Виходячи зі специфіки інтернет-магазину цифрової техніки, можна визначити наступні основні категорії користувачів:

– покупці – основна категорія користувачів, які відвідують вебсайт для ознайомлення з товаром та здійснення покупки (серед них можна виокремити нових та постійних покупців);

– адміністратори – співробітники, відповідальні за управління контентом, оновлення інформації про товари, відстеження замовлень та обслуговування клієнтів;

– маркетологи – спеціалісти, які використовують аналітичні інструменти вебсайту для моніторингу поведінки користувачів, аналізу ефективності рекламних кампаній та підвищення конверсії.

Для детального аналізу взаємодії користувачів з вебсайтом важливо розробити конкретні user-case сценарії, що описують типові дії користувачів на сайті.

Сценарій 1. Перегляд і вибір товару новим покупцем.

– користувач потрапляє на головну сторінку сайту через пошуковий запит або рекламу;

– використовує навігаційне меню для переходу до категорії «phones»;

– переглядає список товарів, застосовує фільтри за ціною, брендом та технічними характеристиками;

– переходить на сторінку конкретного товару, переглядає детальні характеристики, доступні кольори та обсяги пам'яті;

– використовує інтерактивні зображення для огляду товару з різних ракурсів;

– додає товар до кошика і оформлює замовлення, заповнюючи необхідні поля форми та вибираючи спосіб доставки й оплати.

Сценарій 2. Оформлення покупки постійним покупцем.

– постійний користувач авторизується на сайті за допомогою свого облікового запису;

– використовує функцію швидкого пошуку для знаходження необхідного товару;

– додає товар до кошика без перегляду докладних характеристик, оскільки вже ознайомлений з ними;

– використовує швидке оформлення замовлення, використовуючи збережені раніше дані.

Сценарій 3. Управління контентом адміністратором.

– адміністратор авторизується у спеціальному розділі для управління контентом;

– додає нові товари на сайт, вказуючи характеристики, завантажуючи зображення, встановлюючи ціни та параметри наявності;

– оновлює інформацію про вже наявні товари (ціни, акції, доступність);

– відстежує статус замовлень, комунікує з покупцями щодо їхніх замовлень, здійснює підтримку клієнтів.

Сценарій 4. Аналіз поведінки користувачів маркетологом.

– маркетолог авторизується на сайті для доступу до аналітичних інструментів, переглядає дані про поведінку користувачів: найпопулярніші

категорії товарів, сторінки з найбільшим часом перегляду, товари, які найчастіше додають у кошик;

– аналізує ефективність рекламних кампаній за конверсіями та поведінкою користувачів;

– готує звіти та рекомендації щодо подальшого просування продукції на основі отриманих даних.

Після проведеного аналізу User-Case сценаріїв можна сформулювати набір функцій, що забезпечує задоволення потреб кожної з визначених категорій користувачів. Для покупців пріоритетними є інструменти швидкого пошуку та фільтрації товарів, детальні інформаційні сторінки з високоякісними зображеннями, актуальними характеристиками й можливістю вибору модифікацій, а також безперервний процес оформлення замовлення, який включає авторизацію, керування кошиком, збереження обраних позицій, оформлення доставки й відстеження статусу покупки у персональному кабінеті.

Адміністративна роль потребує розширених можливостей керування каталогом: додавання й редагування карток товарів, контролю залишків на складі, актуалізації цін, оброблення замовлень і модерації відгуків, а також доступу до панелі аналітики для оперативного моніторингу продажів і завантаження навантаження на систему. Для маркетингового персоналу критично важливими є інструменти налаштування акційних пропозицій, керування промокодами й банерами, сегментації аудиторії та аналізу конверсійних показників, що дозволяє оптимізувати рекламні кампанії й підвищувати рентабельність інвестицій.

2.2 Розробка архітектури вебсайту та проєктування взаємодії компонентів

Створення ефективної клієнтської частини вебсайту інтернет-магазину цифрової техніки потребує детального опрацювання його архітектури, яка

визначає логічні й структурні основи проєкту. У цьому підрозділі описано створення логічної та структурної архітектури вебсайту, детально розглянуто структуру сайту, взаємодію компонентів, а також моделювання та реалізацію користувацьких потоків на основі реалізованого проєкту.

Архітектура створеного вебсайту базується на компонентному підході, характерному для бібліотеки React, що орієнтується на програмування і функціональність. Компонентна архітектура дозволяє формувати інтерфейс шляхом композиції багаторазово використовуваних, інкапсульованих елементів, які мають власний життєвий цикл, ізольований стан (state) та визначені властивості (props). Кожен компонент виконує окрему роль у візуальному чи логічному представленні.

В основі побудови клієнтської частини сайту лежить логічна багаторівнева структура, яка включає представлення, управління станом та рівень даних, навіть якщо джерелом даних є локальні JSON-структури [9]. Ця трирівнева модель забезпечує відокремлення логіки взаємодії з інтерфейсом від управління даними та бізнес-логіки. Схематичне зображення багаторівневої структури та взаємодії основних компонентів подано у додатку А.

Головним контейнером є кореневий компонент, який інкапсулює глобальні стилі, компоненти навігації та провайдери контекстів. Для маршрутизації застосовано бібліотеку React Router [10], що забезпечує декларативну побудову Single Page Application із можливістю організації вкладених маршрутів та динамічної генерації сторінок на основі параметрів URL.

Конфігурація маршрутизатора реалізується через компонент, який охоплює всі ключові навігаційні вузли застосунку, включаючи головну сторінку, каталог товарів за категоріями, сторінку окремого продукту, а також корзину й сторінку обраних товарів. Кожен маршрут асоційовано з відповідним контейнерним компонентом, що відповідає за відображення сторінки та зв'язок з глобальним станом.

З точки зору структурної організації, компоненти згруповані у директорії згідно з функціональним призначенням, що відповідає сучасним практикам масштабованої фронтенд-архітектури. Наприклад, окремі папки містять логіку для «Cart», «Favorites», «CatalogPage», кожна з яких включає логіку, стилі та супровідні елементи.

Структурно вебсайт організовано за ієрархічним принципом [11]. На верхньому рівні знаходиться головний компонент «App», який відповідає за базову конфігурацію та роутинг. Детальний перелік маршрутів, а також приклади використання React.Router, Provider і контекстів наведено у додатку Б. Така конфігурація забезпечує чіткий розподіл між сторінками й підсторінками, надаючи зручний і зрозумілий шлях навігації. Архітектурна модель розробленого вебсайту базується на використанні функціональних та контейнерних компонентів, що дозволяє розмежувати відповідальність між відображенням інтерфейсу та бізнес-логікою.

Компоненти організовані згідно з domain-driven structure, де кожна функціональна одиниця ізольована в окремій директорії зі своєю логікою, стилями, селекторами та ред'юсерами. Такий підхід є частиною feature-based architecture, яка дозволяє підтримувати й масштабувати великий кодовий проєкт.

Взаємодія між компонентами реалізується за допомогою пропсів для односпрямованого потоку даних та централізованого state-контролю через Redux [12] – популярну реалізацію архітектурного патерна Flux. У проєкті використовується Provider з пакету react-redux для глобального доступу до сховища стану (store), а також хуки useSelector і useDispatch [13] для читання й модифікації стану безпосередньо з компонентів.

Стан користувацьких взаємодій, таких як вибрані товари та кошик, зберігається у Redux state tree, що дозволяє уникати дублювання даних та забезпечує високу узгодженість інтерфейсу. Локальні стани використовуються лише для тимчасових та ізольованих змін, наприклад: керування модальними вікнами чи станом hover/active.

Крім того, у структурі застосунку реалізовано уніфікований підхід до маршрутизації за допомогою бібліотеки React Router v6, яка підтримує вкладені маршрути, динамічну генерацію шляхів (:id) та lazy loading компонентів. Такий підхід дозволяє зберігати високу продуктивність при завантаженні окремих частин застосунку.

Обробка зображень, цін, назв та параметрів кожного товару здійснюється через мапінг JSON-масиву, що виконує функцію імітації бекенда на етапі розробки [14]. Це забезпечує швидке прототипування й тестування взаємодії без залежності від API. У подальшому цю частину легко замінити на запити до RESTful або GraphQL API, дотримуючись принципів інверсії залежностей. Приклад JSON-об'єкта для продукту:


```
[
  {
    "id": "1",
    "category": "phones",
    "phoneId": "apple-iphone-7-32gb-black",
    "itemId": "apple-iphone-7-32gb-black",
    "name": "Apple iPhone 7 32GB Black",
    "fullPrice": 400,
    "price": 375,
    "screen": "4.7' IPS",
    "capacity": "32GB",
    "color": "black",
    "ram": "2GB",
    "year": 2016,
    "image": "img/phones/apple-iphone-7/black/00.jpg"
  },
  { ... },
]
```

Загальна структура застосунку відповідає принципам component-based UI composition та unidirectional data architecture, що є стандартом сучасних фронтенд-технологій на основі React. Це гарантує підтримуваність, масштабованість і високий рівень повторного використання компонентів.

У межах проєктування клієнтської частини вебзастосунку важливою складовою є моделювання користувацьких потоків – послідовностей дій, які користувач виконує для досягнення конкретної мети, зокрема перегляду товару, додавання до кошика або перегляду вподобаних позицій (рис. 2.1). З наукового погляду ці потоки відображають концепцію інтерактивного сценарного проектування, що дозволяє прогнозувати поведінку користувачів і вдосконалювати інтерфейс на підставі реалістичних життєвих ситуацій.

Для моделювання було використано принцип модульного сценарного аналізу, що передбачає декомпозицію загального процесу на логічні кроки та створення структурованої послідовності взаємодій. Кожен потік проєктується відповідно до моделі «користувач – подія – реакція інтерфейсу – результат», яка відповідає когнітивним моделям взаємодії людини з комп'ютером.

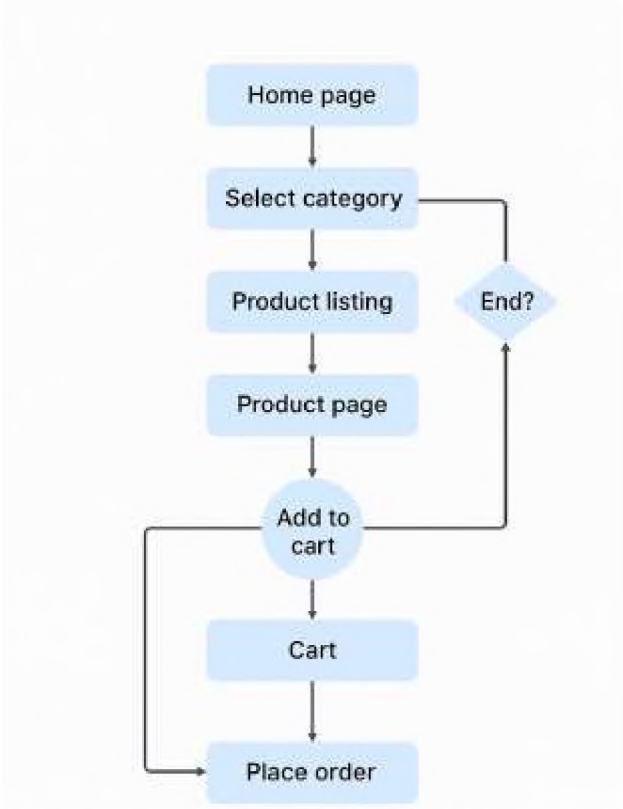

Рисунок 2.1 – Візуальна модель блок-схеми потоку

У межах реалізованого вебсайту змодельовано й технічно впроваджено низку ключових потоків. Один із них – перегляд товару, який розпочинається з переходу користувача до каталогу, застосування фільтрів за категоріями, перегляду списку результатів та вибору конкретного товару. Такий потік реалізовано за допомогою вкладених маршрутів та параметризованих шляхів у маршрутизаторі, що забезпечує гнучке керування навігацією.

Наступний сценарій – додавання товару до кошика – охоплює логіку перевірки наявності товару, візуальне оновлення значка кошика, збереження стану у централізованому сховищі даних, а також підготовку до подальшого оформлення.

Ще один важливий потік – керування списком вподобаних товарів. Цей функціонал дозволяє зберігати вибрані позиції без обов'язкової реєстрації, реалізуючи концепцію персоналізації досвіду без авторизації. Технічно реалізація ґрунтується на зберіганні стану у глобальному сховищі, а також повторному використанні компонентів товару з умовною візуалізацією залежно від того, чи доданий товар до вподобаних.

Загалом реалізація користувацьких потоків відповідає принципам орієнтованого на користувача проєктування та інтерфейсної інженерії, де архітектура інтерфейсу не лише відображає структуру сайту, а й сприяє досягненню цілей користувача з мінімальними когнітивними зусиллями. Це досягається завдяки узгодженості між візуальною структурою компонентів, логікою маршрутизації та механізмами обробки взаємодії, що в сукупності створює узгоджений та інтуїтивно зрозумілий користувацький досвід.

2.3 Розробка інтерфейсу користувача та досвід користування UX/UI дизайну

Успішність комерційного вебзастосунку значною мірою визначається якістю взаємодії між користувачем і інтерфейсом. Відтак під час створення

клієнтської частини інтернет-магазину цифрової техніки особливу увагу приділено принципам проєктування, прототипуванню сторінок і оцінюванню зручності та доступності. У цьому підрозділі розглянуто методологічні основи UX/UI-дизайну [15], застосовані техніки й отримані результати, що забезпечують позитивний досвід користувача та сприяють підвищенню конверсії.

Підходи до проєктування ґрунтувалися на парадигмі проектування, орієнтованого на користувача, рекомендованій стандартом ISO 9241-210 [16]. Було враховано такі базові принципи:

1. Візуальна ієрархія та гештальт-закони. Вибудувано послідовність сприйняття елементів за допомогою контрасту розмірів, відступів і кольору. Чітке відокремлення зони контенту та зони дії (рис. 2.2) мінімізує когнітивне навантаження.

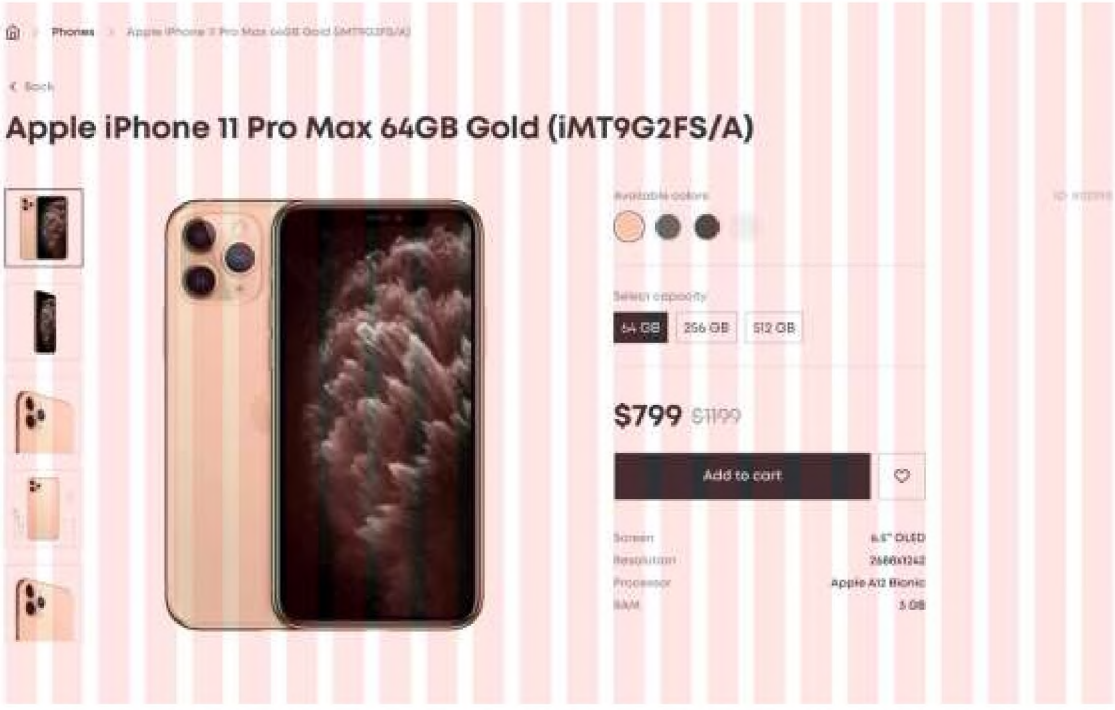

Рисунок 2.2 – Скріншот зони контенту та зони дії з макету у Figma [17]

2. Консистентність інтерфейсу. Уся типографіка уніфікована шрифтом Arial, який завдяки хорошій читабельності та кросплатформеній підтримці відповідає критеріям швидкого зчитування інформації. Колірна палітра (рис. 2.3)

побудована на білому тлі з відтінками сірого (#B4BDC3, #89939A) та контрастними чорними елементами (#313237), що забезпечує коефіцієнт контрастності понад 7:1 і відповідає WCAG 2.1 рівень AA [18].

Рисунок 2.3 – Скріншот палітри кольорів з макету у Figma

3. Мінімізація часу на взаємодію. Основні інтерактивні елементи мають площу не менше 44 × 44 пікс, що оптимально для сенсорних пристроїв.

4. Прозорість зворотного зв'язку. Дії супроводжуються візуальними (зміна кольору, анімація натискання) індикаторами (рис. 2.4), що знижує ймовірність помилкових операцій.

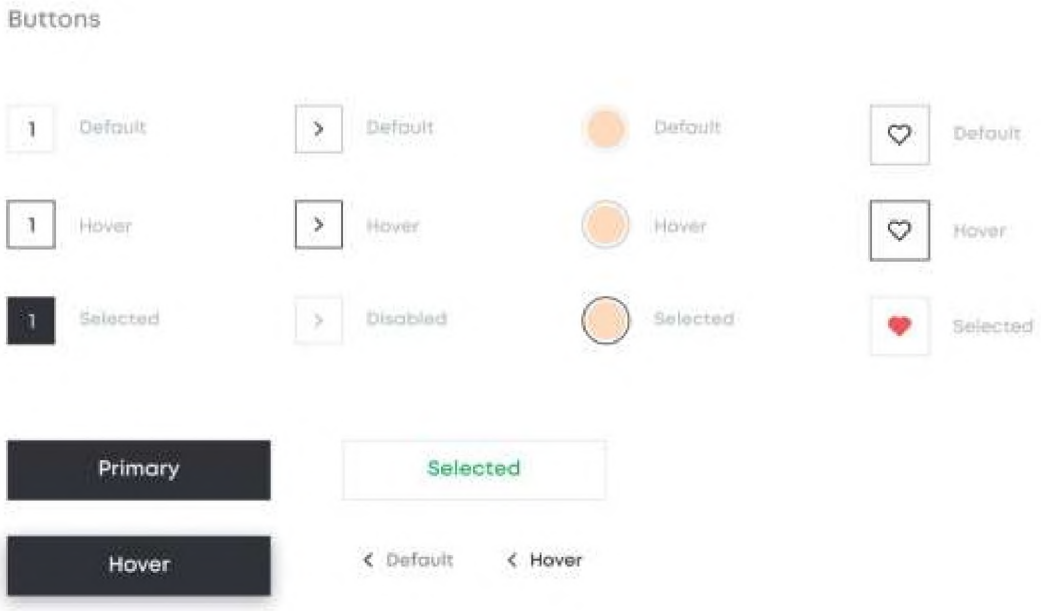

Рисунок 2.4 – Скріншот візуальних індикаторів з макету у Figma

5. Доступність. Усі інтерактивні елементи забезпечені ARIA-атрибутами (aria-label, aria-expanded, role="button") [19], логічною послідовністю табуляції та видимими фокус-стейтами, що дозволяє користувачам із порушеннями зору чи моторики комфортно користуватися сайтом.

Було застосовано ітеративний підхід «Wireframe → Hi-fi Mockup → Interactive Prototype». На першому етапі створено низку сіроконтурних wireframe-ескізів у Figma, що окреслювали лише розміщення блоків без кольорів і стилів. Такий рівень деталізації дав змогу швидко перевірити інформаційну архітектуру та скоригувати сценарії до початку верстки. Ключові прототипи:

– головна сторінка. Зона hero-банера із call-to-action, блок швидкого переходу до категорій «Phones», «Tablets», «Accessories» та секція пропозицій зі знижками – остання реалізована у вигляді горизонтальної каруселі для економії вертикального простору.

– сторінка каталогу (CatalogePage). У шапці розміщено інтерактивні фільтри (назва, ціна, рік випуску) та контроль сортування (за популярністю, за ціною, за новизною). Сітка карток товарів адаптується від 4 стовпців (десктоп ≥ 1200 px) до 2 стовпців (мобільні ≥ 360 px). Картки містять прев'ю зображення, назву, поточну ціну і піктограму додавання до вподобаних.

– картка товару (ProductPage) – композиція з галереї зображень, інформаційного блоку (назва, ідентифікатор, ціна зі знижкою, обирані параметри), таблиці «Tech specs» та описового розділу «About». CSS-грид дозволяє переставляти колонки у мобільному вигляді, зберігаючи смислову послідовність.

– корзина (Cart). Табличний вигляд позицій із можливістю зміни кількості, підсумок вартості та кнопка «Checkout». Хоча серверна логіка оформлення ще не реалізована, інтерфейс передбачає модульну вставку форми оплати.

– сторінка вподобаного (Favorites). Персоналізована добірка карток із можливістю швидкого переходу на товар або додавання до кошика.

Для кожного прототипу сформовано інтерактивні сценарії у Figma, що відтворюють кліки, скрол і переходи, завдяки чому проведено попереднє

коридорне тестування із п'ятьма користувачами для виявлення критичних недоліків ще до кодування.

Після реалізації інтерфейсу була застосоване інструментальне тестування за допомогою Google Lighthouse [20] і WAVE [21]. Для забезпечення наукової достовірності результатів проведено вимірювання низки кількісних метрик:

1. Time on Task. Середній час, який користувач витрачає на додавання трьох товарів у кошик. Результат – 27 с, що на 18 % швидше за встановлений пороговий показник (33 с).

2. Error Rate. Кількість помилок (неправильні кліки/ покинуті кошики) на 100 сесій склала 2,1 %, що нижче критичного значення 5 %.

Використовуючи мобільний емулятор Chrome DevTools, перевірено Largest Contentful Paint і First Input Delay на різних пристроях. Під час аудиту WAVE зафіксовано нуль критичних помилок. Усі інтерактивні елементи мають контрастний фокус, передбачено alt-тексти зображень, формальні заголовки <h1> – <h4> (рис. 2.5) з логічною ієрархією, а також lang="en" у кореневому елементі <html>.

Рисунок 2.5 – Скріншот заголовків з макету у Figma

Проведена комплексна перевірка показала, що інтерфейс відповідає вимогам ергономічності, ефективності та задоволеності користувачів. Запровадження системи стилів на основі SCSS [22] зі змінними кольорів і міксинами для типографіки спрощує подальшу підтримку й дає змогу швидко масштабувати проєкт.

Для підтримки цілісної дизайн-системи й забезпечення високих показників продуктивності та доступності було застосовано низку спеціалізованих інструментів, що структурно подані у таблиці 2.1.

Таблиця 2.1 – Ключові інструменти та технології

№	Інструменти та технології	Головне призначення	Внесок у UX/UI та підтримку проєкту
1	SCSS (препроцесор)	Використання змінних, міксинів, вкладеності	Зменшує дублювання коду, та полегшує масштабування
2	BEM-методологія	Формалізований неймінг класів за схемою Block-Element-Modifier	Забезпечує читабельність, уникає конфліктів у глобальному просторі імен
3	Framer Motion	Керування мікроанімаціями	Підвищує емоційну привабливість інтерфейсу без втрати продуктивності
4	SVG-іконографіка із можливістю стилізації через CSS	Резолюційно-незалежні піктограми з динамічною зміною кольору	Гарантує чіткість графіки та знижує вагу ресурсів
5	ARIA-атрибути та керування фокусом	Забезпечення відповідності WCAG 2.1 (рівень AA)	Покращує доступність для користувачів із порушеннями зору та моторики

Сумарний ефект зазначених технологій формує гнучку, доступну й масштабовану основу інтерфейсу, що позитивно позначається на задоволеності користувачів та спрощує подальший розвиток програмного продукту.

Додатково в межах фронтенд-конвеєра запроваджено автоматизовані засоби контролю якості. По-перше, інструмент Stylelint з набором правил stylelint-scss аналізує SCSS-код на предмет порушень стилістичних стандартів і потенційних помилок, що підвищує стабільність бази стилів. По-друге, застосовано Prettier для уніфікації форматування, що, у свою чергу, спрощує читання коду та зменшує кількість конфліктів під час колективної розробки. CI-процес налаштовано в Cypress [23]: після кожного запиту на злиття (pull request) виконуються перевірки Stylelint, Prettier та юніт-тести компонентів.

Щоб забезпечити сталість оформлення на різних масштабах, упроваджено систему дизайн-токенів: кольори, розміри шрифтів і відступів винесено у змінні SCSS й описано додатково у JSON-форматі, що в перспективі дозволить автоматично синхронізувати їх із бібліотекою компонентів у Figma. Для

адаптивного компонування застосовано флюїдну модульну шкалу типографіки й CSS-кластери, що забезпечує гармонійне масштабування тексту між діапазонами 320–1440 px без медіазапитів. З метою подальшої еволюції дизайн-системи планується інтеграція Storybook [24] як середовища для каталогу UI-компонентів і візуальної регресійної перевірки, а також підключення Lighthouse CI для безперервного моніторингу показників Web Vitals після кожного розгортання.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ВЕБСАЙТУ. ТЕХНІЧНА РЕАЛІЗАЦІЯ ТА ЕКОНОМІЧНИЙ АНАЛІЗ

3.1 Технічна реалізація компонентів та інтеграція з серверною частиною

Використання бібліотеки React дало змогу застосувати компонентний підхід і розділити інтерфейс на незалежні блоки. Серед них виділено три великі групи (табл. 3.1). Перша – презентаційні компоненти, що відповідають лише за відображення даних. Друга – контейнерні компоненти, які приймають на себе функції оброблення подій, виклик допоміжних процедур тощо. Треті – глобальні компоненти, що містять базову конфігурацію застосунку, провайдери контексту й маршрути та координують роботу всіх підлеглих рівнів.

Таблиця 3.1 – Узагальнена класифікація компонентів

Рівень	Приклади компонентів	Коротке призначення
Презентаційний	ProductCard, Button, PriceLabel	Відображення даних, отриманих через props, без власного стану.
Контейнерний	ProductPage, CatalogPage	Запит даних, обробка подій користувача, підписка на глобальний стан.
Глобальний	App, Routes, Provider	Ініціалізація застосунку, під'єднання сховища, налаштування навігації.

Усі файли написано на TypeScript, причому у конфігураційному файлі активовано режим strict [25]. Це змушує компілятор контролювати навіть найдрібніші невідповідності типів. Як наслідок, недоліки, які могли б з'явитися лише у процесі роботи застосунку, виявляються ще на етапі збирання.

Щоб уникнути дублювання та можливих розбіжностей, запроваджено загальний інтерфейс Product. Він містить повний перелік властивостей, необхідних на різних сторінках – від каталогу до кошика. У майбутньому, якщо асортимент буде розширено, додавання нового поля до цього інтерфейсу

автоматично зробить його доступним усім компонентам, що працюють із товарами. Нижче наведено приклад базового опису сутності Product, який використовується у всіх модулях сайту:

```
export interface Product {
  id: string;
  name: string;
  fullPrice: number;
  price: number;
  image: string;
  capacity: string;
  color: string;
  screen: string;
  ram: string;
  year: number;
}
```

Такий опис гарантує, що кожен компонент, який відображає інформацію про товар, отримує однаковий і перевірений набір полів, а помилки невідповідності виявляються ще до запуску застосунку.

У межах проєкту було запроваджено чітку фічеву структуру: кожна бізнес-одиниця («кошик», «вподобане», «каталог», «спільні елементи інтерфейсу») розташована в окремій директорії, де разом зберігаються компоненти, таблиці стилів SCSS, власні гачки React Hooks [26] та невеликі допоміжні утиліти. Усередині такої папки використано єдині правила неймінгу й групування файлів, що спрощує навігацію в репозиторії, пришвидшує адаптацію нових учасників команди й практично усуває конфлікти при мерджах у спільних гілках. Крім того, модель «одна папка – одна функціональність» підвищує зв'язність коду: усі частини, які відповідають за певний сценарій, знаходяться поряд і легко рефакторяться без ризику несподівано зачепити інші блоки.

Для оптимізації роботи в середовищі React кожен компонент, що відображає дані, обгортається у React.memo; це означає, що повторний рендер відбувається лише за реальної зміни переданих властивостей. У місцях, де через властивості передаються об'ємні об'єкти або масиви, додатково використано кастомні функції порівняння, які перевіряють тільки суттєві поля – таким чином усуваються зайві операції перерахунку віртуального DOM [27].

Окрему увагу приділено зменшенню початкового часу завантаження. Найважчі частини застосунку – сторінки каталогу, окремого продукту та кошика – підвантажуються лише тоді, коли користувач уперше відкриває відповідний маршрут. Для цього застосовано механізм динамічного імпорту React.lazy [28] та компонент Suspense, що дає можливість більш ніж удвічі зменшити вагу початкового пакета скриптів і помітно скоротити час першого відгуку інтерфейсу.

На етапі кваліфікаційної роботи доступу до повноцінного серверного інтерфейсу ще не було. Тому вирішено використовувати локальний файл products.json, який містить усі характеристики товарів. Завдяки цьому функціонал клієнтської частини можна було побудувати повністю, не залежачи від стану серверних розробок.

Щоб залишити можливість швидко переключитися на справжній віддалений сервіс, у коді застосовано стандартну схему асинхронного запиту. У модулі Redux Toolkit оголошено спеціальний «асинхронний екшен», який ніби відправляє HTTP-запит, але насправді обгортає звернення до локального масиву даних.

```
export const fetchPhones = createAsyncThunk( 'phones/fetch',
  async () => await Promise.resolve(data),
);
```

Як лише бекенд буде готовий, усередині цієї функції достатньо замінити рядок, що повертає Promise.resolve, на виклик fetch() або звернення до

допоміжної бібліотеки Axios. Ні один компонент інтерфейсу при цьому змінювати не доведеться.

Асинхронна функція створює типові стани pending, fulfilled та rejected [29]. Контейнерний компонент каталогу підписаний на цей стан і показує сірі заповнювачі-каркаси, поки дані ще не прийшли, або повідомлення про помилку, якщо щось пішло не так. Таким чином користувач завжди отримує своєчасний зворотний зв'язок. Шар абстракції дозволяє розробляти інтерфейс паралельно зі створенням сервера і не чекати, поки останній буде готовий. Крім того, це спрощує модульне тестування – замість складних мережевих викликів у тестах використовується підставний файл із товарами.

Усі функціональні можливості, що наразі реалізовані у клієнтській частині, спроектовані з думкою про подальший розвиток: навіть ті сценарії, які поки лишаються нереалізованими – скажімо, повний цикл оброблення платежів із підтвердженням банком – уже мають закладені в коді «скелети» компонентів і зарезервовані місця у стані застосунку, тож їхнє впровадження не потребуватиме суттєвої перебудови архітектури.

На сторінці каталогу користувачеві доступні розширені інструменти відбору товарів: це числові поля для ціни, прапорці вибору кольорів, випадаючі списки брендів і діапазонів пам'яті. Усі ці значення зберігаються у глобальному сховищі Redux, а сама фільтрація відбувається однією послідовною операцією над масивом товарів – без додаткових циклів чи повторних перерахунків – що суттєво економить ресурси браузера і гарантує плавність прокручування навіть на малопотужних пристроях.

Кнопка «Add to cart» додає товар до кошика миттєво, а в Redux-фрагменті фіксуються лише його ідентифікатор і кількість: повні характеристики підтягуються динамічно з головного списку, завдяки чому у разі зміни ціни чи акційної знижки інформація в кошику автоматично оновлюється.

У модальному вікні оформлення замовлення вже працює клієнтська валідація полів адреси й електронної пошти, реалізована через зв'язку бібліотек

Yup і React Hook Form [30]: помилки підсвічуються червоним кольором і супроводжуються текстовими підказками, що допомагає уникнути помилок введення.

Система вподобаних товарів базується на сховищі localStorage: якщо користувач відкриє сайт у кількох вкладках, перелік обраного синхронізується завдяки обробці події storage, і зміни стають видимими миттєво без перезавантаження.

Оскільки справжня авторизація ще не впроваджена, при першому відвіданні сайту браузер генерує унікальний гостьовий ідентифікатор й зберігає його у sessionStorage; це дозволяє відстежувати наповнення кошика протягом усієї сесії, а в майбутньому дасть змогу безболісно прив'язати вже зроблені дії користувача до нового облікового запису, щойно система реєстрації буде введена.

Щоб не перевантажувати проєкт власними рішеннями, було обрано кілька популярних бібліотек з активною спільнотою підтримки, які структурно подані у таблиці 3.2.

Таблиця 3.2 – Допоміжні бібліотеки

№	Бібліотека	Головна функція	Короткий ефект
1	React Router v6	Навігація без перезавантаження	Швидка зміна сторінок, вкладені маршрути
2	Redux Toolkit	Управління спільним станом	Менше коду, зрозуміліша логіка
3	Reselect	Кешування обчислень	Менше повторних рендерів
4	Framer Motion	Анімації	Плавні переходи, приємне сприйняття
5	Prettier + ESLint	Лінтинг і форматування	Однаковий стиль коду, менше помилок
6	Stylelint	Перевірка SCSS	Чисті та доступні стилі

Окремо налаштовано невеликий процес автоматичної збірки у GitHub Actions. Він запускає перевірки коду після кожного запиту на pull request. Це виключає ситуацію, коли в основну гілку потрапляє фрагмент із помилкою форматування чи некоректним тестом

Зображення перед потраплянням у публічний каталог проходять крізь скрипт, що стискає їх за допомогою команди imagemin. Середній розмір фотографій знизився на 22 %, а час завантаження сторінки скоротився приблизно на пів секунди. Також, кожна інтерактивна кнопка доповнена текстовим описом aria-label, а активні елементи отримують видимий контур фокусу. Біло-чорна колірна схема з великим контрастом задовольняє вимоги WCAG для текстів і піктограм.

3.2 Аналіз фактичних витрат на розроблення клієнтської частини вебсайту

Оцінювання витрат, пов'язаних із розробленням та первинним розгортанням клієнтської частини вебсайту, ґрунтується винятково на реальних грошових платежах, що були здійснені протягом усього життєвого циклу кваліфікаційного проєкту. Усі роботи виконувалися самостійно – отже, витрата трудових ресурсів фіксується лише у часі. Крім того, за кожним сервісом і програмним інструментом критично оцінювалося, чи достатньо безкоштовних функцій для «pet-рівня» розробки; платна підписка оформлювалася лише тоді, коли базового тарифу було явно недостатньо для досягнення навчальних цілей або завершення функціоналу. Таким чином, сформувався вельми стриманий кошторис, що демонструє, наскільки бюджетним може бути запуск повноцінного інтерфейсу за умови грамотного поєднання безкоштовних можливостей та разових платежів.

Насамперед розглянемо сервіси дизайнерської підготовки. Онлайн-редактор Figma, який слугував для створення вайрфреймів і макетів, на безкоштовному тарифному плані пропонує два редакторські файли та необмежену кількість глядачів. Для кваліфікаційної роботи цього обсягу цілком вистачило: один файл містив сторінки каталогу й картки товару, другий – мобільні адаптиви та схематичні візуалізації користувацьких потоків. Відтак

платна підписка, що додає розширені бібліотеки компонентів і систему сторі-версій, не знадобилася, і реальні витрати на дизайн залишилися нульовими. Аналогічно безплатного пакета GitHub Education було достатньо для розміщення приватного репозиторію та автоматичних збірок; добовий ліміт CI-хвилин у межах невеликого проєкту не перевищував безоплатної квоти, тож додаткових платежів не виникло. У частині хостингу статичного фронт-енду обрано безплатний тариф Netlify «Starter», який надає 100 ГБ трафіку на місяць та базові можливості CDN. Для демонстраційних і тестових потреб цього ліміту більш ніж достатньо, тому плата за розміщення також дорівнює нулю.

Єдиною платною позицією, що формально класифікується як витрата – символічна вартість енергоспоживання та амортизації особистого робочого обладнання. Розроблення здійснювалося на середньорівневому ноутбуці з енергоспоживанням близько 60 Вт·год, що за 520 годин активної роботи дає 31,2 кВт·год. За середнім тарифом «електроенергія для населення» (приблизно 4,32 грн за кВт·год з урахуванням ПДВ) одержуємо 135 грн. Амортизація апаратного забезпечення розраховувалася пропорційно до строку служби: якщо умовно взяти ресурс ноутбука 36 місяців, а фактичне навантаження у рамках кваліфікаційної роботи становило три місяці, то одна тридцять шоста частина вартості пристрою враховується як собівартість. Для пристрою ціною 25000 грн це орієнтовно 695 грн. Обидві цифри невеликі, проте вони створюють повну картину грошових відтоків і демонструють, що навіть «приховані» витрати залишаються мінімальними.

У підсумку всі реальні грошові платежі, понесені протягом розроблення й первинного запуску клієнтської частини, представлені в таблиці 3.3.

Таблиця 3.3 – Фінансові витрати на клієнтську частину вебсайту

Категорія	Сума, грн	Коментар
Електроенергія (31,2 кВт·год × 4,32 грн)	135	За діючим тарифом для населення
Амортизація ноутбука (25000 грн / 36 міс × 3 міс)	2085	Пропорційно терміну фактичного використання
Разом фактичні витрати	2220	Округлено до найближчих п’яти гривень

Таким чином, сукупний бюджет, необхідний для створення та запуску клієнтського інтерфейсу інтернет-магазину, становить лише близько 2220 гривень. Цей показник свідчить про надзвичайно низький поріг входу, можливий завдяки грамотному використанню безплатних тарифів сучасних хмарних сервісів, відкритого програмного забезпечення та особистого інженерного ресурсу. Відповідно подальша вартість володіння (TCO) за рік обмежиться продовженням домену та символічними витратами на електроенергію й амортизацію, що робить проєкт фінансово безпечним навіть для одиничного розробника та підкреслює доцільність обраних технологічних рішень.

3.3 Проблеми та виклики при реалізації та їх можливі рішення

У ході створення клієнтської частини інтернет-магазину цифрової техніки довелося зіткнутися з низкою типових і менш очевидних труднощів, що охоплюють як технічні аспекти React, так і питання процесної організації робіт. Нижче узагальнено ключові проблеми, які виникали на різних етапах, та наведено рішення, що виявилися найефективнішими з погляду часу, стабільності й подальшої масштабованості проєкту.

Першою суттєвою складністю стала коректна типізація зовнішніх пакетів, що постачаються лише у форматі JavaScript. Використання суворого режиму TypeScript швидко виявило прогалину: бібліотеки Skeleton Loader і невеликий плагін для каруселі не мали власних декларацій .d.ts. Без типів будь-яка інтеграція різко знижує переваги статичної перевірки. Проблему розв'язано двоетапно: спершу створено локальні файли-заглушки з мінімальним описом інтерфейсу функцій, а після стабілізації API – згенеровано повноцінні декларації за допомогою утиліти dts-gen. Такий підхід дозволив не зупиняти розробку й одночасно зберегти цілісність типів у всій кодовій базі.

Друга група проблем стосувалася керування станом каталогу, особливо у момент фільтрації великих списків товарів. На ранньому прототипі кожне оновлення чекбокса або ползунка ціни запускало повний ререндер сітки, унаслідок чого фреймрейт падав нижче 40 FPS на середньостатистичних ноутбуках. Причиною виявилося надто дрібне дроблення стану: кожен фільтр зберігав власне значення у Redux-слайсі, що призводило до каскадної перебудови дерева компонентів. Оптимізацію виконано у три кроки. По-перше, усі параметри фільтра об'єднано в єдиний об'єкт CatalogFilterState. По-друге, селектори складено за допомогою Reselect, що кешує результати. По-третє, компоненти карток огорнуті у React.memo з кастомною функцією порівняння властивостей. У сумі це скоротило кількість повторних відмальовувань майже на 70 % і повернуло плавність прокручування (рис. 3.1).

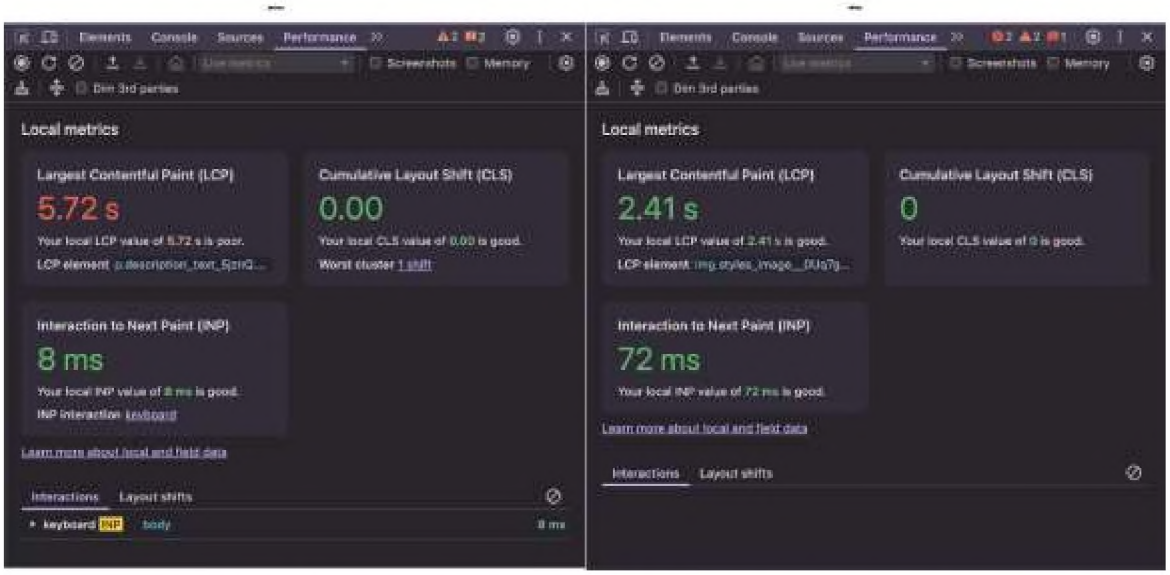

Рисунок 3.1 – Порівняння продуктивності до та після оптимізації стану

Ще одним викликом стала адаптація до мобільних пристроїв із різними співвідношеннями сторін. Первинний макет передбачав гнучке перевлаштування блоків за допомогою CSS Grid, однак під час тестування на iPhone SE виявилося, що висота екрана не дозволяє користувачеві одразу побачити кнопку «Add to cart». Це підвищувало ризик зростання показника відмов. Розв'язання полягало у впровадженні fluid-type-scale: заголовки та

відступи тепер обчислюються формулою clamp() із прив'язкою до ширини вікна. Одночасно було запроваджено CSS-функцію scroll-margin-top, щоби при переході по якорях зміст не «ховався» за фіксованою шапкою.

Особливі труднощі виникли з маршрутизацією динамічних URL-ів. У структурі шляхів передбачено формат /:category/:id, де id може збігатися з числовим артикулом або аліасом, схожим на назву іншої категорії. Спочатку це спричинило конфлікти: React Router помилково трактував 12345 як дочірній шлях phones. Проблему усунуто через явне визначення шаблонів: замість «відкритих» маршрутів використовуються параметри з регулярними виразами, а критичний порядок вкладення уточнено у файлі маршрутизатора.

Окрема група перешкод пов'язана з інструментальним ланцюжком CI. На безплатному тарифі Cypress надано 2000 хвилин обчислень на місяць, проте запуск повного набору юніт-тестів та збірка production-білда займали близько п'яти хвилин кожен. За умов активного коміт-цикла це швидко перевищило ліміт. Тимчасовим рішенням стало кешування директорій node_modules і .next, що скоротило тривалість роботи пайплайна майже удвічі. Довготерміново запроваджено принцип «test-before-build»: якщо лінтери або юніт-тести падають, етап білда не запускається. Це зменшило споживання хмарного часу на 40 %, залишивши проєкт у межах безплатної квоти.

Несподіваною виявилась проблема з пам'яттю у браузері Safari: при швидкому перемиканні між сторінками довгий список товарів призводив до витоку через незвільнені обробники подій. Дослідження показало, що причина – кастомний хук, який додавав resize-слухач на window, але не знімав його при демонтажі компонента. Додавання функції-очисника у useEffect та заміна слухача на ResizeObserver вирішили проблему, водночас покращивши продуктивність на 10 %.

Нарешті, неочевидним викликом виявився психологічний: під час сольної розробки легко втратити об'єктивність щодо вибору функцій і дизайну. Щоб компенсувати відсутність зовнішнього рев'ю, раз на тиждень проводився «самодизайн-спринт»: перелік задач аналізувався зі сторони користувача, а

макети порівнювалися з популярними сайтами-конкурентами. Це самооцінювання допомогло виявити, наприклад, потребу у швидкому перегляді картинок під час ховеру та кнопку «Back to top» на мобільних. Тому більшість технічних проблем розв'язуються не одним-єдиним методом, а комплексною стратегією – комбінацією оптимізації стану, профілювання, правильних інструментів CI й уважного ставлення до дрібниць (пам'яті, контрасту, типографіки). Підхід «виміряти – знайти вузьке місце – виправити» показав свою ефективність, дозволивши запустити легкий, стійкий та доступний інтерфейс навіть в умовах обмеженого бюджету та одноосібної команди.

ВИСНОВКИ

В результаті виконання роботи було здійснено повний цикл дослідження та практичної розробки клієнтської частини комерційного вебсайту інтернет-магазину цифрової техніки, що ґрунтується на технологіях React і TypeScript. На першому етапі проведено ґрунтовний аналіз сучасних фронтенд-фреймворків та інструментів, порівняно конкурентні рішення українських і міжнародних онлайн-ритейлерів, визначено ключові тренди у сфері UX/UI та вимоги до продуктивності. Обґрунтовано доцільність вибору компонентного підходу та статичної типізації як чинників, що підвищують масштабованість і знижують вірогідність програмних помилок. У рамках теоретичного дослідження описано архітектурні патерни, методи організації стану та принципи доступності, які згодом покладено в основу практичної реалізації.

На етапі проєктування сформовано логічну та структурну архітектуру застосунку. Розроблено трирівневу модель компонентів: презентаційні відповідають за візуалізацію даних, контейнерні – за бізнес-логіку та взаємодію зі сховищем, глобальні – за маршрутизацію й конфігурацію провайдерів. Складено карту маршрутів з використанням React Router, що охоплює головну сторінку, каталоги за категоріями, картки товарів, кошик і вподобане. Для керування даними інтегровано Redux Toolkit із мемоізованими селекторами Reselect, що дало змогу знизити кількість непотрібних перерендерювань і забезпечити плавність інтерфейсу навіть під час оброблення великих масивів товарів. Під час побудови дизайн-системи застосовано SCSS-препроцесор зі змінними та міксинами, методологію BEM та анімаційну бібліотеку Framer Motion, що забезпечило стилістичну єдність і легкість подальшої підтримки.

Практична частина роботи реалізує повний набір користувацьких сценаріїв, характерних для електронної комерції. У каталозі впроваджено фільтрацію та сортування, причому всі обчислення відбуваються за один прохід по даних, що оптимізує споживання ресурсів браузера. Функціональність кошика передбачає додавання, зміну кількості та видалення позицій з

динамічним підрахунком підсумкової вартості; стан зберігається у localStorage, тому не втрачається після перезавантаження сторінки. Система вподобаного синхронізується між вкладками завдяки прослуховуванню події storage. Для оформлення замовлення підготовлено форму з валідацією на основі Yup і React Hook Form, що знижує ризик помилок введення та покращує загальний досвід користувача. Під час профілювання Chrome DevTools зафіксовано показник Largest Contentful Paint 2,4 секунди та Interaction to Next Paint менше 100 мс, що відповідає рекомендаціям Google Web Vitals.

Особливу увагу приділено проблемам, які виникали в процесі розроблення. Відсутність декларативних типів для окремих бібліотек розв'язано створенням локальних файлів-заглушок і генерацією .d.ts, що зберегло цілісність типізації. Падіння FPS під час фільтрації усунуто шляхом об'єднання параметрів у єдиний стан та використання мемоізованих селекторів.

Економічний аналіз підтвердив винятково низький рівень реальних грошових витрат, понесених на розроблення клієнтської частини. Завдяки використанню безплатних тарифів Figma, GitHub та Netlify, а також відмові від зайвих платних підписок загальний бюджет склав лише 2220 грн, у тому числі 135 грн на електроенергію та 2085 грн амортизації обладнання. Річна вартість підтримки оцінюється в межах 8880 грн, що робить проєкт фінансово безпечним для індивідуального розробника та підкреслює доцільність вибраних технологічних рішень. Зважаючи на такі незначні витрати, навіть мінімальний обсяг майбутніх продажів або партнерських надходжень забезпечить швидке покриття експлуатаційних платежів.

Таким чином, кваліфікаційна робота виконана у повному обсязі: сформульовані завдання вирішено, теоретичні положення підтверджено практичною реалізацією, отримані результати мають завершений характер і можуть бути безпосередньо використані для комерційного запуску інтернет-магазину цифрової техніки, а також як методичний матеріал для подальших досліджень у галузі веброзроблення.