Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка e-commerce платформи для комп'ютерних ігор. Частина 1. Архітектура клієнта»

Виконав: студент __IV__ курсу
групи __5ПІ-21б__
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Рельке А. А.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Бабюк Н. П.
(прізвище та ініціали)
Рецензент:. к.т.н., доц. каф ОТ Савицька Л.А.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ _____
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
"21" березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рельке Артуру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи – розробка e-commerce платформи для комп'ютерних ігор. Частина 1. Архітектура клієнта

Керівник роботи Бабюк Наталія Петрівна, к.т.н., доцент кафедри ПЗ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "20" березня 2025 р. №97.

2. Термін подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: архітектура клієнтської частини e-commerce платформи - до 15 компонентів (каталог, кошик, навігація, пошук тощо), функціональні вимоги - до 25, продукти в каталозі - до 100000, швидкість завантаження - до 1.5 сек, час відгуку - до 200 мс, адаптивний дизайн 320px-4K, сумісність з браузерами не старше 2 років, JavaScript бандл - до 500KB gzip.

4. Зміст текстової частини: вступ, аналіз e-commerce платформ для ігор, огляд клієнтських технологій, теоретичні основи React архітектури, алгоритми оптимізації каталогу, розробка клієнтських модулів, навігація та маршрутизація, управління кошиком, адаптивний інтерфейс, покращення продуктивності, віртуалізація та кешування, тестування модулів, інтеграція компонентів, висновки, додатки.

5. Перелік графічного матеріалу: мета та задачі роботи, порівняння e-commerce платформ, архітектурна схема клієнтської частини, блок-схема алгоритмів оптимізації каталогу, діаграма React компонентів, схема маршрутизації, результати тестування продуктивності, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Бабюк Н. П., к.т.н., доцент кафедри ПЗ	24.03.25р	03.06.25р

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану e-commerce платформи для комп'ютерних ігор	24.03.25-30.03.25	вик
2	Аналіз аналогів e-commerce платформи для комп'ютерних ігор	02.04.25-16.04.25	вик
3	Розробка алгоритмів та модулів архітектури клієнта e-commerce платформи	16.04.25-21.04.25	вик
4	Тестування архітектури клієнта e-commerce платформи	22.04.25-17.05.24	вик
5	Оформлення матеріалів до захисту БКР	24.03.25-30.05.25	вик

Студент _____ Рельке А.А.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Бабюк Н. П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 00404

Рельке А. А. Розробка e-commerce платформи для комп’ютерних ігор. Частина 1. Архітектура клієнта: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 58 с.

На укр. мові. Бібліогр.: 24 назв.; рис. : 18; табл. : 2.

У бакалаврській кваліфікаційній роботі було розроблено архітектуру клієнтської частини e-commerce платформи для комп'ютерних ігор з використанням сучасних веб-технологій React та супутніх фреймворків. Проведено аналіз предметної галузі e-commerce платформ для цифрових ігрових продуктів та обґрунтовано необхідність розробки дієвої клієнтської архітектури для забезпечення швидкодії, зручності використання та конкурентоспроможності на ринку цифрових розваг. Запропоновано компонентну архітектуру системи, що забезпечує оптимальне відображення каталогу продуктів, управління користувацьким інтерфейсом та взаємодію з серверною частиною.

Розроблено метод оптимізації відображення каталогу цифрових продуктів на основі адаптивного завантаження контенту, метод компонентної архітектури для e-commerce систем з автономним управлінням станом та метод інтелектуального кешування даних продуктів на основі аналізу поведінки користувачів.

Створено алгоритми відображення списків продуктів, компоненти деталізованого перегляду товарів, систему навігації та маршрутизації, модулі управління кошиком покупок та адаптивний користувацький інтерфейс.

Проведене тестування підтвердило високу продуктивність та стабільність розробленої архітектури клієнтської частини e-commerce платформи. Система

демонструє швидке завантаження, відсутність критичних помилок та дієвого використання ресурсів браузера.

Ключові слова: e-commerce платформа, комп'ютерні ігри, клієнтська архітектура, React, користувацький інтерфейс, каталог продуктів, веб-технології, компонентна архітектура, цифрова комерція, продуктивність.

ABSTRACT

UDC 00404

Relke A. A. Development of an E-commerce Platform for Computer Games. Part 1. Client Architecture: Bachelor's Qualification Thesis in the specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2024. 58 p.

In Ukrainian. Bibliography: 24 titles; figures: 18; tables: 2.

In the bachelor's thesis, a client-side architecture for an e-commerce platform for computer games was developed using modern web technologies like React and accompanying frameworks. The research included an analysis of the e-commerce domain for digital gaming products and justified the need for creating an effective client-side architecture to ensure high performance, usability, and competitiveness in the digital entertainment market. A component-based system architecture was proposed, enabling optimal display of the product catalog, user interface management, and interaction with the server-side.

A method for optimizing the display of the digital product catalog based on adaptive content loading was developed, as well as a component architecture approach for e-commerce systems with autonomous state management and a method for intelligent caching of product data based on user behavior analysis.

Algorithms for displaying product lists, components for detailed product views, a navigation and routing system, shopping cart management modules, and an adaptive user interface were created.

Testing confirmed the high performance and stability of the developed client-side architecture for the e-commerce platform. The system demonstrates fast loading, no critical errors, and efficient use of browser resources.

Keywords: e-commerce platform, computer games, client architecture, React, user interface, product catalog, web technologies, component architecture, digital commerce, performance.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження

Сучасний ринок цифрових технологій характеризується стрімким зростанням популярності комп'ютерних ігор та потребою у спеціалізованих платформах для їх розповсюдження. Індустрія ігор стала однією з найбільш прибуткових галузей інформаційних технологій, що вимагає створення дієвих e-commerce платформ для задоволення потреб як розробників, так і споживачів.

Традиційні методи продажу фізичних копій комп'ютерних ігор поступово втрачають актуальність, поступаючись місцем цифровим платформам розповсюдження. У цьому контексті розробка спеціалізованої e-commerce платформи для комп'ютерних ігор з використанням сучасних веб-технологій відкриває нові можливості для дієвої організації продажів, управління каталогом продуктів та забезпечення якісного користувацького досвіду [1].

Автоматизовані системи електронної комерції є найдієвишими для управління великими каталогами цифрових продуктів, забезпечують миттєвий доступ до контенту, точно відображають актуальну інформацію про продукти та підлягають налаштуванню для різних категорій користувачів. E-commerce платформа для комп'ютерних ігор є багатофункціональним інструментом для користувачів, який дозволяє істотно знизити витрати на традиційні методи дистрибуції порівняно з існуючими підходами.

Швидкодія та зручність користувацького інтерфейсу при роботі з каталогом ігор є критичними для забезпечення конкурентоспроможності та залучення цільової аудиторії. У сфері цифрових розваг це має особливе значення, оскільки якісний пользувацький досвід безпосередньо впливає на конверсію та лояльність клієнтів.

Наприклад, у випадках коли існує необхідність у швидкому пошуку та придбанні конкретних ігрових продуктів, дієва архітектура клієнтської частини є ключовим елементом для забезпечення задоволеності користувачів. Повільна робота інтерфейсу, складна навігація або неінтуїтивна структура каталогу

можуть стати факторами відмови від покупки та переходу до конкурентних платформ. Оптимізована клієнтська частина допомагає користувачам дієво знаходити потрібні продукти, порівнювати їх характеристики та здійснювати покупки без технічних перешкод.

Наразі існує обмежена кількість спеціалізованих програмних платформ, які дозволяють комплексно вирішувати задачі створення e-commerce платформ для ігрової індустрії з використанням сучасних веб-технологій. Наявні аналоги часто мають недостатню гнучкість архітектури, обмежені можливості кастомізації та низьку продуктивність клієнтської частини, що негативно впливає на користувацький досвід.

Тому розробка архітектури клієнтської частини e-commerce платформи для комп'ютерних ігор з використанням сучасних фреймворків та технологій є актуальною задачею, яка має високу практичну значимість для ігрової індустрії.

Зв'язок роботи з науковими програмами, планами, темами

Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження

Метою дослідження є підвищення дієвості функціонування e-commerce платформ для комп'ютерних ігор за рахунок розробки оптимізованої архітектури клієнтської частини, що забезпечує швидкодію, зручність використання та масштабованість системи.

Відповідно до поставленої мети в бакалаврській кваліфікаційній роботі потрібно вирішити такі задачі:

- аналіз предметної галузі e-commerce платформ для комп'ютерних ігор;
- розробка архітектури клієнтських модулів системи управління продуктами;
- розробка алгоритмів відображення каталогу та деталей продуктів;
- розробка компонентів навігації та маршрутизації сторінок;
- розробка модулів управління кошиком покупок;
- розробка адаптивного користувацького інтерфейсу;

- розробка системи оптимізації продуктивності клієнтської частини;
- тестування розроблених програмних модулів;
- інтеграція компонентів у єдину архітектуру клієнтської частини.

Об'єкт дослідження

Процес функціонування клієнтської частини e-commerce платформ для комп'ютерних ігор з використанням сучасних веб-технологій.

Предмет дослідження

Методи та засоби розробки архітектури клієнтських модулів e-commerce платформи для комп'ютерних ігор з використанням React та супутніх технологій.

Методи дослідження

У процесі дослідження застосовувалися: теорія архітектури програмного забезпечення; методи проєктування клієнт-серверних додатків; алгоритми оптимізації продуктивності веб-інтерфейсів; методи юзабіліті-тестування; технології фронтенд розробки, зокрема React для створення компонентної архітектури, сучасні підходи до управління станом додатку та маршрутизації; комп'ютерне моделювання для аналізу та перевірки архітектурних платформ.

Наукова новизна отриманих результатів

1. Вперше запропоновано метод динамічного управління навігаційними переходами в e-commerce системах, особливість якого полягає у комбінуванні реактивного програмування з адаптивною валідацією контенту та каскадним оновленням залежних компонентів, що забезпечує плавні переходи між розділами платформи та покращує користувацький досвід взаємодії з інтерфейсом.

2. Подальшого розвитку отримав метод асинхронного отримання даних з серверної частини, особливість якого полягає у використанні React Hooks з інтегрованою валідацією структури даних та обробкою помилок через try-catch конструкції, що дозволяє забезпечити надійність додатку та реактивне оновлення інтерфейсу при зміні даних.

3. Вперше запропоновано метод маршрутизації на основі керування станом компонента, особливість якого полягає у відмові від традиційних

бібліотек маршрутизації на користь внутрішнього управління станами selectedProductId та content з використанням тернарних операторів для умовного рендерингу, що спрощує архітектуру системи та покращує продуктивність для додатків з обмеженою кількістю основних розділів.

4. Подальшого розвитку отримав метод асинхронної взаємодії з серверним API для модифікації стану кошика покупок, особливість якого полягає у використанні async/await синтаксису з фабричним методом API-клієнта та централізованою обробкою помилок, що забезпечує єдиний підхід до комунікації з серверною частиною та спрощує підтримку і масштабування системи.

Практичне значення отриманих результатів

Практичне значення отриманих результатів полягає в тому, що на основі теоретичних досліджень розроблено архітектуру клієнтської частини e-commerce платформи для комп'ютерних ігор, яка може бути впроваджена в різних сферах цифрової комерції – від невеликих інді-студій до великих ігрових корпорацій.

Особистий внесок здобувача

Усі наукові результати, викладені у кваліфікаційній роботі, отримані автором особисто, отримані автором особисто. Комплексна бакалаврська кваліфікаційна робота виконана у співпраці з Дмитрієв В.Г.

Апробація матеріалів бакалаврської кваліфікаційної роботи

Результати кваліфікаційної роботи доповідалися на всеукраїнській науково-практичній конференції «Технології створення e-comerce платформ» (2025), та всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Застосування алгоритмів машинного навчання для оптимізації веб-додатків» (2024).

Публікації

Результати кваліфікаційної роботи доповідалися на конференціях Міжнародна науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи», ВНТУ, 2025. (МН-2024) та Всеукраїнської науково-практичної конференції молодих (МН-2025).

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз стану технологій e-commerce платформ для комп’ютерних ігор

Ринок цифрової дистрибуції комп'ютерних ігор демонструє стабільне зростання, досягнувши у 2024 році обсягу понад 180 мільярдів доларів. Цифрові продажі становлять близько 85% загального обсягу продажів ігор, що свідчить про фундаментальні зміни в способах поширення та споживання ігрового контенту. За прогнозами аналітиків, цей тренд посилюватиметься, і до 2027 року частка цифрових продажів може досягти 93%.

Така трансформація пов'язана із загальною діджиталізацією, розвитком високошвидкісного інтернету та зміною споживчих звичок нових поколінь гравців, які надають перевагу миттєвому доступу до контенту без фізичних носіїв.

Сучасні e-commerce платформи для ігор базуються на складних архітектурних рішеннях. Мікросервісна архітектура забезпечує гнучкість, масштабованість та стійкість платформи через розподіл функціональності на незалежні сервіси. Технології контейнеризації, такі як Docker та Kubernetes, дозволяють дієво управляти розгортанням та масштабуванням окремих компонентів системи. Хмарна інфраструктура, побудована на базі AWS, Azure або Google Cloud, забезпечує високу доступність сервісів та їх адаптацію до змінного навантаження, особливо під час великих релізів чи сезонних розпродажів [2].

На стороні клієнта домінують сучасні JavaScript-фреймворки, як-от React, Angular та Vue.js, що дозволяють створювати інтерактивні та динамічні інтерфейси користувача. Технології крос-платформної розробки, зокрема Electron та React Native, забезпечують єдиний користувацький досвід на різних пристроях – від персональних комп'ютерів до мобільних телефонів. Progressive Web Apps стають все більш поширеними, пропонуючи веб-додатки з досвідом, наближеним до нативних програм. Технології WebGL та WebAssembly

розширюють можливості браузерів, дозволяючи впроваджувати високопродуктивні візуальні ефекти та попередній перегляд ігор безпосередньо на веб-сторінках платформ.

Аналіз сучасного стану технологій e-commerce платформ для комп'ютерних ігор виявляє суттєві можливості для інновацій та вдосконалення. Розробка власної платформи є обґрунтованою з огляду на кілька ключових факторів.

Технологічна трансформація ринку створює унікальне вікно можливостей для впровадження інноваційних підходів. Перехід до нових моделей дистрибуції та споживання контенту відкриває простір для платформ, які краще відповідають сучасним очікуванням користувачів та розробників. Існуючі технологічні обмеження можуть бути подолані за допомогою сучасних фреймворків, таких як React для клієнтської частини та Electron для забезпечення крос-платформності.

Зміни в споживчій поведінці нових поколінь гравців також підтверджують необхідність розробки нових платформ. Сучасні користувачі мають інші очікування щодо інтерфейсів, соціальних функцій та моделей монетизації, які часто не повністю задовольняються існуючими платформами. Власна платформа може бути спроектована з урахуванням цих змін, забезпечуючи більш інтуїтивний та сучасний користувацький досвід [3].

Сучасні веб-технології відкривають значні можливості для технологічних інновацій. React та Electron дозволяють створювати продуктивні, адаптивні та зручні інтерфейси, які можуть забезпечити конкурентну перевагу на ринку. Ці технології дозволяють швидко впроваджувати нові функції та експериментувати з різними підходами до користувацького інтерфейсу.

Отже аналіз сучасного стану технологій e-commerce платформ для комп'ютерних ігор свідчить про динамічний розвиток цієї галузі та наявність суттєвих можливостей для інновацій. Технологічні тренди, включаючи хмарний геймінг, штучний інтелект, блокчейн та розширені соціальні функції, формують нове покоління платформ цифрової дистрибуції.

Водночас існують значні технологічні виклики, пов'язані з масштабуванням, захистом контенту, багатоплатформністю та обробкою даних. Ці виклики створюють можливості для розробки нових, більш дієвих методів.

1.2 Аналіз методів розв’язання задачі

Розробка програмних модулів клієнтської архітектури e-commerce платформи для комп'ютерних ігор вимагає детального аналізу існуючих підходів з урахуванням специфіки галузі, вимог до продуктивності системи та особливостей взаємодії користувачів з платформою. Ключовим аспектом при проектуванні такої системи є вибір оптимальної архітектури клієнтської частини, яка забезпечить швидкий і зручний доступ до цифрових товарів, персоналізовану взаємодію та високу швидкодію.

В контексті розробки e-commerce платформи для комп'ютерних ігор із використанням фреймворків React та Electron, можна розглянути декілька архітектурних підходів. Одним із найпоширеніших є компонентно-орієнтована архітектура, яка надає розподіл інтерфейсу на незалежні, повторно використовувані компоненти. React, як бібліотека для побудови користувацьких інтерфейсів, повністю відповідає цьому підходу, забезпечуючи однонаправлений потік даних та дієвий механізм оновлення DOM через Virtual DOM. Цей архітектурний підхід дозволяє створювати модульні, масштабовані рішення з чітким розмежуванням відповідальності між компонентами системи.

Альтернативним підходом є використання архітектури Flux або її похідних (зокрема, Redux), які передбачають централізоване управління станом додатку. Даний підхід вирішує складності управління станом у великих додатках шляхом впровадження однонаправленого потоку даних та єдиного джерела істини. В контексті e-commerce платформи це особливо важливо для реалізації таких функцій як кошик покупок, система персоналізованих рекомендацій, відстеження історії покупок та управління цифровими ліцензіями [4].

При розробці кросплатформного додатку з використанням Electron постає питання інтеграції веб-орієнтованих технологій з можливостями десктопних

застосунків. Electron надає доступ до нативних API операційної системи через Node.js, що дозволяє реалізувати функціонал, недоступний у веб-середовищі. Для дієвої взаємодії між процесами рендерингу (фронтенд на React) та основним процесом (backend на Node.js) можна використовувати архітектурний патерн «міст», який забезпечує слабке зв'язування між компонентами різних рівнів.

З точки зору організації коду клієнтської частини, доцільно застосувати патерн «атомарний дизайн», який надає розподіл компонентів за рівнями складності: атоми (базові елементи інтерфейсу), молекули (групи атомів), організми (функціональні блоки), шаблони та сторінки. Такий підхід забезпечує високу повторюваність використання компонентів, уніфікацію інтерфейсу та спрощує підтримку стилів [5].

Для управління асинхронними операціями, такими як завантаження каталогу ігор, обробка платежів чи інтеграція з зовнішніми сервісами, доцільно використовувати патерн «спостерігач» у поєднанні з реактивним програмуванням. Бібліотеки на кшталт RxJS дозволяють дієво обробляти потоки подій та управляти складною асинхронною логікою.

Враховуючи специфіку e-commerce платформи для комп'ютерних ігор, особливу увагу слід приділити оптимізації продуктивності. Використання технік мемоізації, ледачого завантаження компонентів та оптимізації рендерингу дозволить забезпечити швидку роботу інтерфейсу навіть при відображенні великих каталогів товарів з багатим медіа-контентом.

На основі проведеного аналізу для розробки клієнтської частини e-commerce платформи для комп'ютерних ігор рекомендується використовувати компонентно-орієнтовану архітектуру з централізованим управлінням станом на базі Redux. Додаток буде структуровано відповідно до принципів атомарного дизайну з чітким розмежуванням відповідальності між компонентами різних рівнів. Електрон забезпечить кросплатформну реалізацію з доступом до нативних можливостей операційної системи, а React дозволить створити гнучкий та продуктивний користувацький інтерфейс. Така архітектура забезпечить

баланс між швидкістю розробки, підтримуваністю коду та можливістю масштабування функціоналу платформи в майбутньому.

1.3 Аналіз аналогів

Сучасний ринок цифрової дистрибуції комп'ютерних ігор зображений кількома потужними гравцями, кожен з яких має свої технологічні особливості, переваги та недоліки. Для обґрунтування доцільності розробки власної e-commerce платформи необхідно провести детальний аналіз існуючих аналогів, виявити їхні сильні та слабкі сторони, а також визначити потенційні напрями для інновацій та вдосконалень. До найбільш відомих платформ відносять:

- Steam;
- Microsoft Store;
- Epic games;
- GOG.

Steam, розроблений Valve Corporation, залишається беззаперечним лідером ринку з часткою близько 75% у сегменті PC-ігор та понад 120 мільйонів активних користувачів щомісяця. Платформа пропонує найбільшу бібліотеку ігор (понад 50 000 найменувань) та розвинену екосистему додаткових сервісів.

Steam побудований на власному движку, який розроблявся протягом багатьох років. Клієнтський застосунок написаний з використанням C++ та власних технологій Valve. Платформа використовує гібридну архітектуру з елементами клієнт-серверної та P2P моделей, особливо для дистрибуції контенту. Система Steam Workshop забезпечує механізми для створення та поширення користувацького контенту (рис. 1.1). Платформа також включає власну систему DRM (Steam DRM), яка інтегрується з іграми [6].

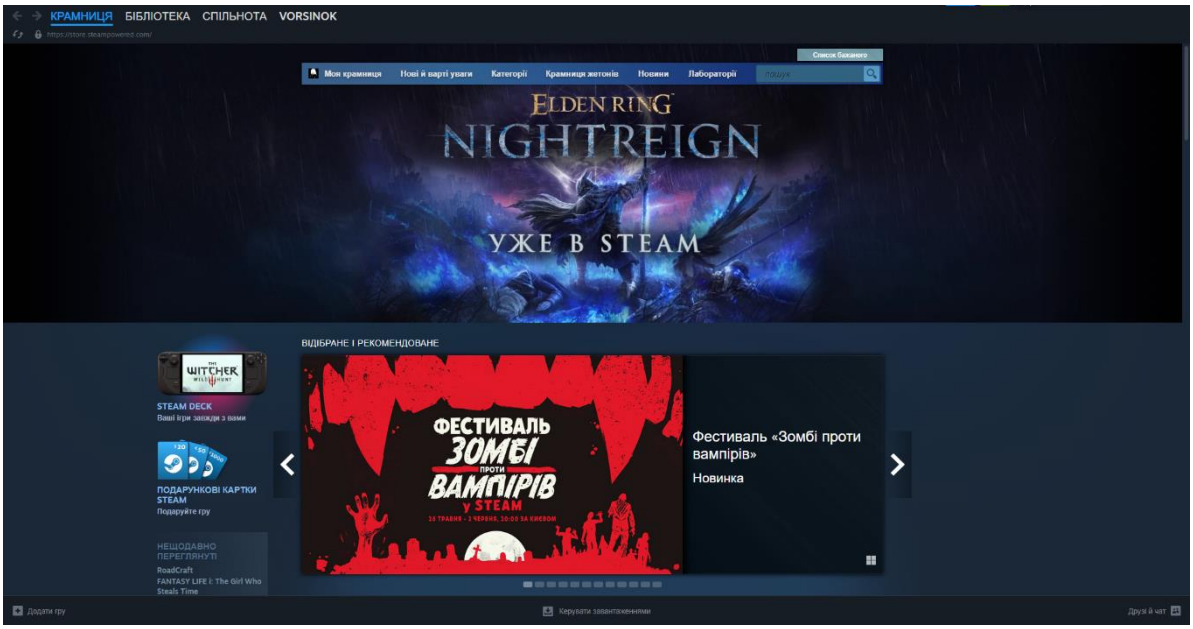

Рисунок 1.1 – Приклад роботи застосунка Steam

Steam пропонує розвинену систему соціальних функцій, включаючи профілі користувачів, досягнення, спільноти та систему рекомендацій. Платформа забезпечує автоматичні оновлення ігор та хмарне збереження прогресу. Інтеграція зі сторонніми сервісами та підтримка модифікацій створює додаткову цінність для користувачів. Система знижок та розпродажів Steam стала галузевим стандартом.

Незважаючи на оновлення інтерфейсу у 2019 році, користувацький досвід Steam часто критикується за надмірну складність та застарілий дизайн. Висока комісія для розробників (30%) створює бар'єри для входу малих студій. Система відкриття нових ігор недостатньо дієва, що призводить до "поховання" багатьох якісних проєктів під масою релізів. Обмежена підтримка мобільних платформ та відсутність повноцінного веб-інтерфейсу для покупок і управління бібліотекою також є суттєвими недоліками [7].

Epic Games Store (EGS) запущений у 2018 році як прямий конкурент Steam, використовуючи фінансові ресурси від успіху Fortnite для агресивного просування через ексклюзивні релізи та щотижневі безкоштовні ігри.

EGS побудований на технологіях компанії Epic, включаючи елементи Unreal Engine. Клієнтський застосунок розроблений з використанням веб-

технологій (Electron) та C++. Платформа інтегрована з Unreal Engine для спрощення процесу публікації ігор, розроблених на цьому движку. EGS використовує власну систему онлайн-сервісів Epic Online Services для авторизації, досягнень та крос-платформного мультиплеєру [8].

EGS пропонує нижчу комісію для розробників (12% проти 30% у Steam), що робить платформу привабливою для видавців. Регулярні безкоштовні ігри залучають нових користувачів та розширюють базу активних гравців. Інтеграція з Unreal Engine спрощує робочий процес для розробників, що використовують цей движок. Платформа має сучасний, мінімалістичний інтерфейс з нижчим споживанням ресурсів порівняно зі Steam (рис. 1.2).

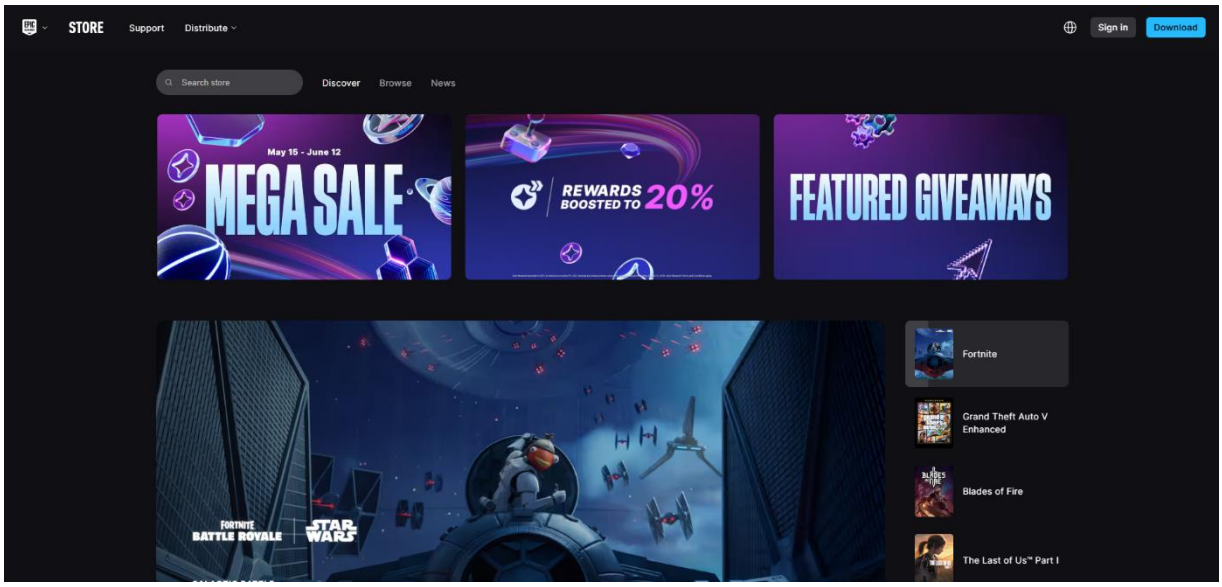

Рисунок 1.2 – Приклад роботи застосунка Epic games

EGS критикується за брак базової функціональності, такої як кошик для покупок, користувацькі відгуки, форуми та досягнення (хоча деякі з цих функцій поступово впроваджуються). Обмежена бібліотека ігор порівняно з конкурентами зменшує привабливість платформи для користувачів. Політика ексклюзивності викликає негативну реакцію в частини ігрової спільноти. Відсутність дієвої системи відкриття контенту ускладнює знаходження нових релізів, особливо від незалежних розробників [9].

GOG, що належить CD Projekt, позиціонується як платформа з філософією «без DRM» та фокусом на збереженні класичних ігор, адаптованих для сучасних систем.

GOG Galaxy 2.0, клієнт платформи, розроблений з використанням C++ та сучасних веб-технологій [7]. Особливість платформи – відсутність обов'язкового DRM, що дозволяє користувачам повністю володіти придбаними іграми без обмежень. GOG інвестує значні ресурси в адаптацію класичних ігор для сучасних операційних систем. Платформа також пропонує унікальну функцію інтеграції бібліотек з інших платформ в єдиний інтерфейс GOG Galaxy (рис. 1.3).

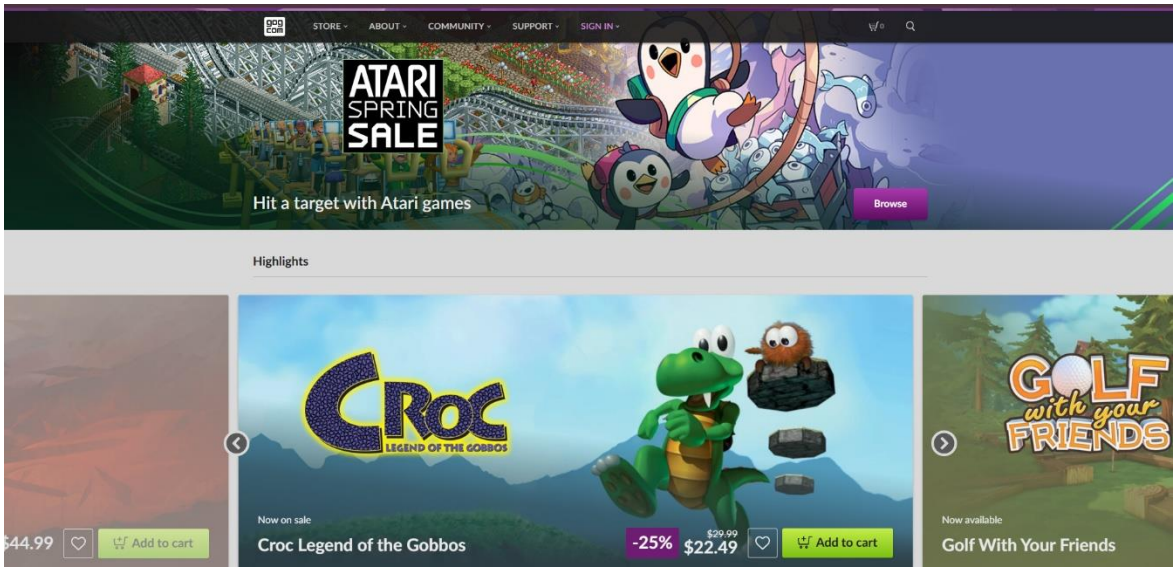

Рисунок 1.3 – Приклад роботи застосунка GOG

Відсутність DRM є ключовою перевагою GOG, що забезпечує повну власність користувачів над придбаними іграми та можливість грати офлайн без обмежень. Платформа пропонує високу якість обслуговування клієнтів та регіональну цінову політику. Функція інтеграції всіх ігрових бібліотек в GOG Galaxy 2.0 створює додаткову цінність для користувачів з іграми на різних платформах. GOG також відомий своєю політикою повернення коштів та бонусами лояльності [10].

Обмежена бібліотека сучасних AAA-релізів зменшує привабливість платформи для користувачів, які цікавляться новинками. Відсутність підтримки

модифікацій та користувацького контенту обмежує довгострокову цінність придбаних ігор. Менша база користувачів порівняно з конкурентами призводить до обмеженої соціальної функціональності. Відсутність DRM, хоча і є перевагою для користувачів, створює бар'єри для залучення деяких великих видавців.

Microsoft активно розвиває свою екосистему на перетині PC та консольного ринку, з особливим фокусом на сервіс підписки Xbox Game Pass.

Microsoft Store інтегрований безпосередньо в операційну систему Windows 10/11 та використовує Universal Windows Platform (UWP) для розповсюдження додатків. Xbox Game Pass реалізує модель доступу за підпискою до ротаційної бібліотеки ігор (рис. 1.4). Платформа підтримує технологію Xbox Play Anywhere, що дозволяє грати в придбані ігри як на Xbox, так і на PC. Microsoft також розвиває сервіс хмарного геймінгу Xbox Cloud Gaming, інтегрований в екосистему [11].

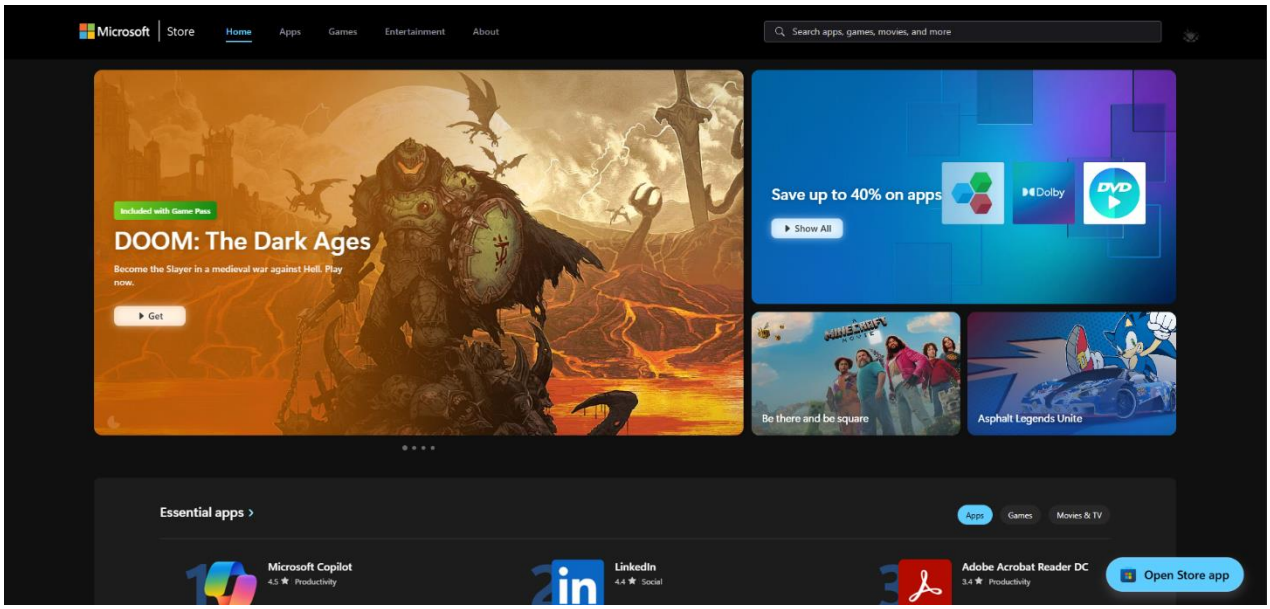

Рисунок 1.3 – Приклад роботи застосунка Microsoft Store

Xbox Game Pass пропонує високу цінність за вартість підписки, надаючи доступ до сотень ігор, включаючи преміум-релізи в день виходу. Інтеграція з екосистемою Xbox створює безперешкодний досвід між PC та консолями. Сервіс хмарного геймінгу дозволяє грати без потужного обладнання на різних

пристроях. Microsoft активно розвиває функції кросплатформного мультиплеєру та крос-прогресії [12].

Microsoft Store критикується за проблеми з надійністю, швидкістю та зручністю користувацького інтерфейсу. Обмеження UWP-формату створюють технічні складності для модифікацій та сторонніх інструментів. Модель ротації ігор у Game Pass означає, що доступ до конкретної гри може бути тимчасовим. Платформа пропонує обмежені соціальні функції порівняно зі спеціалізованими ігровими платформами [13].

Результати порівняльного аналізу функціональних можливостей розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Steam	Epic Games Store	GOG	Microsoft Store
Технологічний стек	C++, власний движок	Electron, C++	C++, веб-технології	UWP, .NET
Комісія для розробників	30% (знижується до 20%)	12%	30%	30% (12% для ігор на PC)
Крос-платформність	Windows, macOS, Linux	Windows, macOS	Windows, macOS	Windows, Xbox
Мобільна підтримка	Обмежена	Обмежена	Відсутня	Обмежена
Система відкриття ігор	Складна, алгоритмічна	Курована	Курована	Обмежена

Продовження таблиці 1.1

Критерій	Steam	Epic Games Store	GOG	Microsoft Store
Інтеграція з блокчейн	Відсутня	Обмежена	Відсутня	Відсутня
Інтеграція з соцмережами	Обмежена	Базова	Середня	Обмежена
Підтримка інді-розробників	Середня	Висока	Висока	Низька
Персоналізація контенту	Середня	Базова	Низька	Низька
DRM	Обов'язковий	Опціональний	Відсутній	Обов'язковий

Проведений порівняльний аналіз існуючих e-commerce платформ для комп'ютерних ігор виявив значні можливості для інновацій та вдосконалень у цій галузі. Кожна з розглянутих платформ має свої сильні сторони та обмеження, але жодна не пропонує оптимального поєднання всіх критично важливих аспектів для сучасного ринку цифрової дистрибуції ігор.

Розробка власної платформи з використанням сучасних технологій React та Electron зображує обґрунтовану стратегію для створення конкурентоспроможного рішення, здатного заповнити виявлені прогалини на ринку. Технологічний стек React/Electron забезпечить необхідну гнучкість, крос-платформність та швидкість розробки, дозволяючи оперативно адаптуватися до змін ринку та впроваджувати інноваційні функції [14].

Ключовими конкурентними перевагами власної платформи можуть стати: справедлива модель комісій для розробників, справжня крос-платформність, інноваційні підходи до відкриття контенту та інтеграція з новітніми технологіями, такими як блокчейн та хмарний геймінг. Особливо перспективним

напрямом є фокус на підтримку незалежних розробників та створення дієвих механізмів для просування якісних нішевих проєктів, що часто губляться на великих платформах [15].

Водночас, розробка власної платформи пов'язана з суттєвими технологічними викликами та ринковими ризиками, які потребують ретельного планування та поетапного впровадження. Однак, при правильному виконанні, розробка власної e-commerce платформи для комп'ютерних ігор з використанням React та Electron може створити життєздатну альтернативу існуючим рішенням та забезпечити унікальну цінність для розробників і гравців.

1.4 Постановка задач дослідження

Проаналізувавши існуючі рішення у сфері електронної комерції комп'ютерних ігор та враховуючи особливості клієнтської архітектури з використанням React та Electron, було визначено наступні задачі, які необхідно виконати в рамках кваліфікаційної роботи:

- розробити модуль каталогу ігрових продуктів, що забезпечуватиме пошук, фільтрацію та зображення детальної інформації про комп'ютерні ігри з підтримкою різних методів сортування та категоризації;
- розробити модуль управління обліковим записом користувача, який забезпечить реєстрацію, авторизацію, редагування персональних даних та збереження історії покупок;
- розробити модуль кошика та оформлення замовлення, що дозволятиме користувачам додавати товари до кошика, застосовувати промокоди, обирати методи оплати та отримувати цифрові ключі;
- розробити модуль бібліотеки придбаних ігор, який забезпечить зручний доступ до придбаних продуктів, відстеження ігрового прогресу та управління цифровими ліцензіями;
- розробити модуль десктопного клієнту на базі Electron, який забезпечить завантаження, встановлення та оновлення придбаних ігор, а також запуск ігор із локального пристрою;

- розробити модуль комунікації з серверною частиною для синхронізації даних між веб-версією та десктопним клієнтом;
- провести тестування розроблених модулів на відповідність функціональним вимогам, кросплатформність та продуктивність.

1.5 Висновки

Отже, було проведено комплексний аналіз методів розробки клієнтської архітектури e-commerce платформи для комп'ютерних ігор із використанням фрейм

Порівняльний аналіз існуючих e-commerce платформ виявив значні можливості для інновацій у галузі цифрової дистрибуції ігор. Жодна з розглянутих платформ не пропонує оптимального поєднання всіх важливих аспектів сучасного ринку, що підтверджує актуальність розробки власного рішення. Обрана технологічна комбінація React/Electron забезпечує необхідну гнучкість, кросплатформність та швидкість розробки, що дозволить оперативно адаптуватися до динамічних змін ринку.

На основі проведеного аналізу було чітко сформульовано задачі проєкту, що охоплюють усі ключові аспекти функціонування сучасної e-commerce платформи для комп'ютерних ігор: від каталогізації та зображення ігрових продуктів до забезпечення безпечних платежів та управління цифровими ліцензіями. Особлива увага приділяється інтеграції веб та десктопних компонентів системи, що дозволить користувачам отримати єдиний цілісний досвід взаємодії з платформою незалежно від пристрою.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

2.1 Розробка методу виводу деталей про продукт

Метод виводу деталей про продукт є критично важливим компонентом архітектури клієнтської частини e-commerce платформи для комп'ютерних ігор. Його основним призначенням є забезпечення користувачів повною та актуальною інформацією про обрані продукти через дієве взаємодію з серверною частиною системи [16].

Процес отримання та відображення деталей продукту починається з ініціалізації користувачем запиту на перегляд конкретного продукту. На початковому етапі система реєструє намір користувача переглянути детальну інформацію про обраний товар, що активує послідовність операцій для отримання відповідних даних. Цей етап характеризується встановленням контексту запиту та підготовкою системи до обробки інформаційного запиту.

Наступним кроком у реалізації методу є отримання ідентифікатора продукту з параметрів запиту. Система аналізує вхідні дані для визначення конкретного продукту, деталі якого необхідно відобразити. Цей процес включає валідацію отриманого ідентифікатора та його підготовку для подальшого використання в API-запитах. Правильна обробка ідентифікатора продукту є фундаментальною основою для коректного функціонування всього методу.

Третій етап надає встановлення типу дії для API-запиту. Система конфігурує параметр action зі значенням 'products', що вказує серверній частині на необхідність обробки запиту як запиту на отримання інформації про продукти. Це забезпечує правильну маршрутизацію запиту та активацію відповідних обробників на серверній стороні системи.

Центральним компонентом методу є виконання GET-запиту до API для отримання детальної інформації про продукт. На цьому етапі клієнтська частина формує HTTP-запит з відповідними параметрами та передає його до серверної частини системи. Запит містить всі необхідні дані для ідентифікації конкретного продукту та отримання його повної характеристики. Система використовує

асинхронну модель взаємодії для забезпечення оптимальної продуктивності та відгуку користувацького інтерфейсу (рис. 2.1).

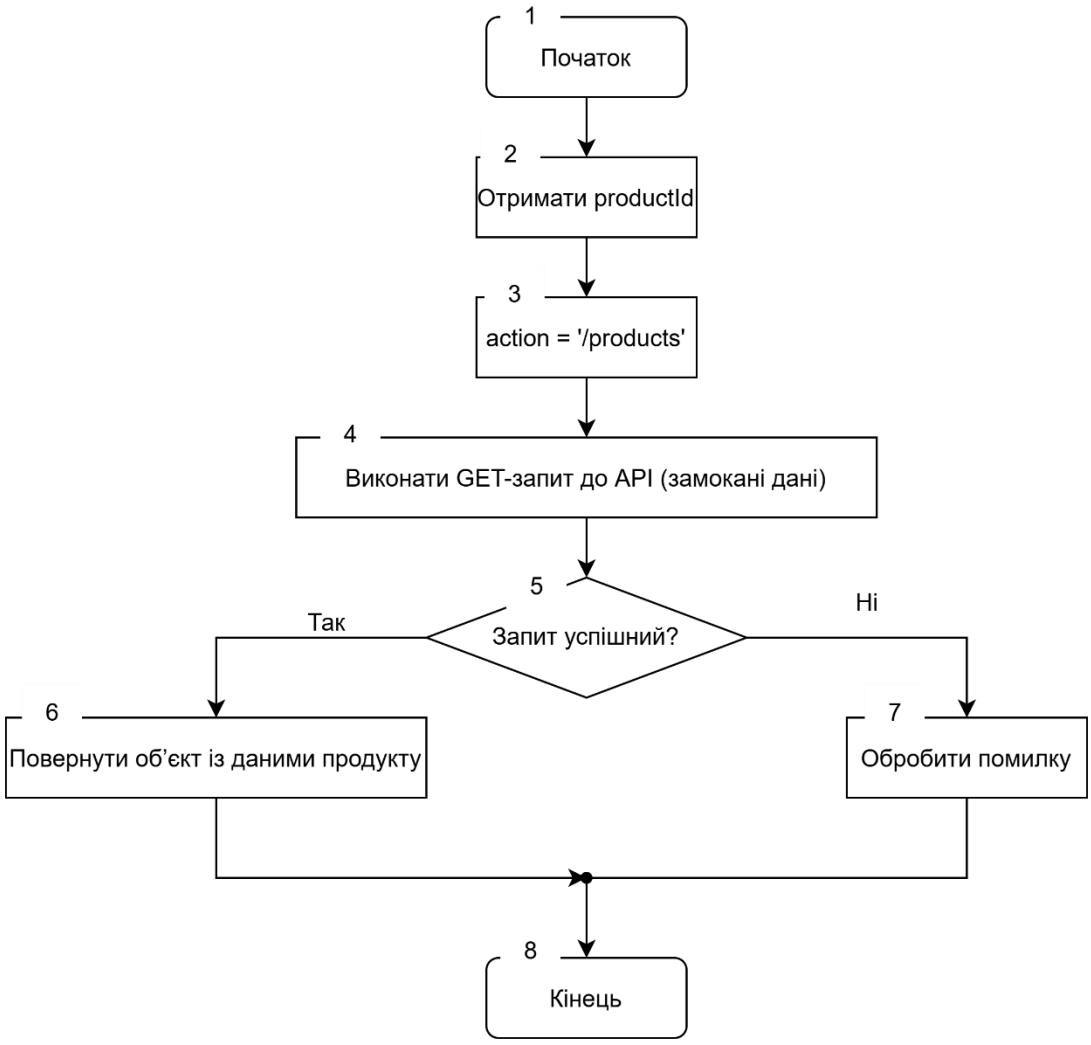

Рисунок 2.1 – Блок-схема методу виводу деталей про продукт

Критично важливим аспектом методу є обробка результатів API-запиту та аналіз успішності операції. Система здійснює перевірку статусу отриманої відповіді для визначення подальшого алгоритму дій. У разі успішного виконання запиту система переходить до етапу обробки отриманих даних та їх підготовки для відображення користувачу. Отримана інформація включає всі необхідні деталі про продукт, такі як назва, опис, ціна, скріншоти, системні вимоги та інші релевантні характеристики.

Процес успішної обробки даних надає створення об'єкта з детальною інформацією про продукт та його інтеграцію в структуру компонента для подальшого відображення. Система забезпечує оптимальне форматування даних відповідно до вимог користувацького інтерфейсу та стандартів відображення інформації про ігрові продукти. Це включає підготовку мультимедійного контенту, форматування текстових описів та структурування технічних характеристик.

У випадку неуспішного виконання API-запиту система активує механізм обробки помилок. Цей процес включає аналіз типу помилки, логування інциденту та відображення відповідного повідомлення користувачу. Система забезпечує graceful degradation, дозволяючи користувачу продовжити взаємодію з платформою навіть у разі тимчасових технічних несправностей із отриманням деталей конкретного продукту.

Завершальним етапом виконання методу є фіналізація процесу та підготовка системи до обробки наступних користувацьких запитів. Система очищує тимчасові ресурси, оновлює стан компонентів та забезпечує готовність до обробки подальших взаємодій користувача з платформою.

Архітектурно метод реалізований з використанням принципів модульності та розділення відповідальностей. Він інтегрується з іншими компонентами клієнтської частини системи через чітко визначені інтерфейси та забезпечує високий рівень надійності та масштабованості. Метод підтримує кешування даних для оптимізації продуктивності та зменшення навантаження на серверну частину системи при повторних запитах на ту саму інформацію про продукт.

2.2 Розробка методу виводу списків всіх продуктів

Метод виводу списків всіх продуктів становить фундаментальну частину архітектури клієнтської системи e-commerce платформи для комп'ютерних ігор. Цей компонент забезпечує користувачам можливість перегляду повного каталогу доступних ігрових продуктів через дієву взаємодію з серверними API-сервісами та надійну обробку отриманих даних.

Ініціалізація процесу отримання списку продуктів розпочинається з активації відповідного запиту в системі. На початковому етапі система встановлює контекст операції та підготовлює необхідні ресурси для взаємодії з серверною частиною платформи. Цей етап характеризується налаштуванням параметрів запиту та ініціалізацією структур даних, необхідних для подальшої обробки інформації про продукти (рис. 2.2).

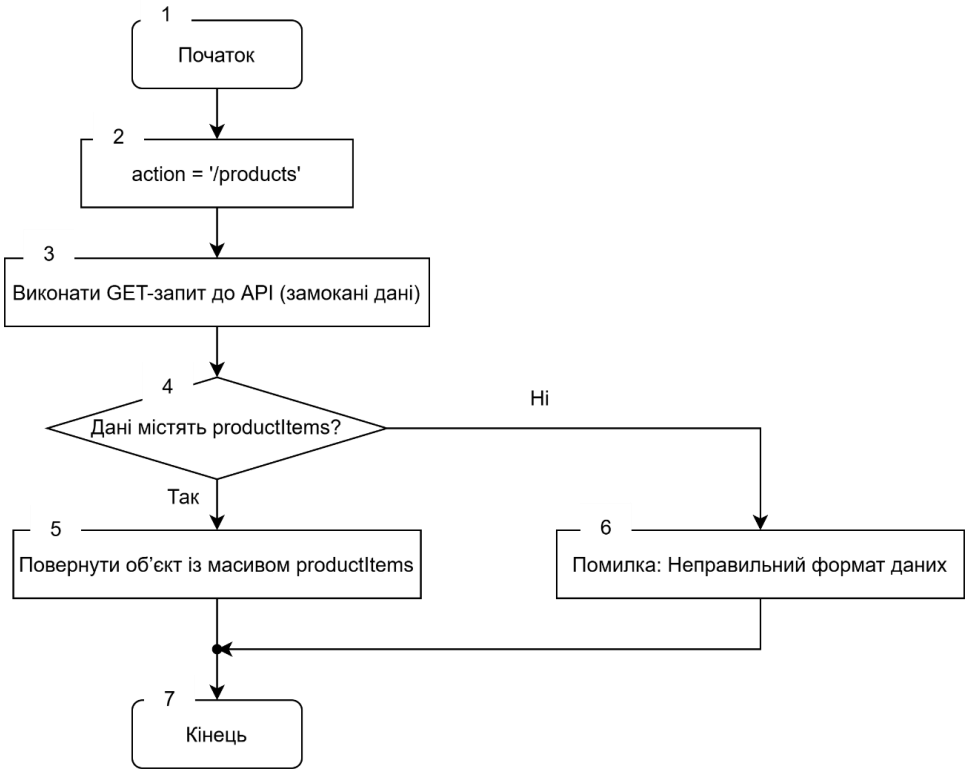

Рисунок 2.2 – Блок-схема методу виводу списків всіх продуктів

Наступним критично важливим кроком є конфігурація типу дії для API-запиту. Система встановлює параметр action зі значенням 'products', що забезпечує правильну ідентифікацію запиту на серверній стороні та активацію відповідних обробників для роботи з каталогом продуктів. Це налаштування гарантує, що серверна частина системи коректно інтерпретує намір клієнта отримати повний перелік доступних ігрових продуктів.

Центральним елементом методу є виконання GET-запиту до API для отримання масиву всіх продуктів платформи. Система формує HTTP-запит з відповідними параметрами та передає його до серверної частини через

встановлені канали комунікації. Запит оптимізований для дієвого отримання великих обсягів даних та підтримує механізми кешування для покращення продуктивності системи. Асинхронна природа взаємодії забезпечує збереження відзивчості користувацького інтерфейсу під час обробки запиту.

Ключовим аспектом архітектури методу є реалізація надійної системи валідації отриманих даних. Після отримання відповіді від API система здійснює комплексну перевірку структури та змісту даних для забезпечення їх відповідності очікуваному формату. Процес валідації включає аналіз наявності масиву productItems у структурі відповіді, що є критично важливим для коректного функціонування подальших етапів обробки.

У випадку успішної валідації система переходить до етапу обробки масиву продуктів. Отримані дані структуруються та адаптуються для оптимального відображення в користувацькому інтерфейсі. Система забезпечує дієве управління пам'яттю та оптимізацію структур даних для швидкого доступу до інформації про окремі продукти. Процес включає форматування основних характеристик продуктів, таких як назви, ціни, категорії та превью-зображення, відповідно до вимог компонентів інтерфейсу.

Архітектура методу надає комплексний механізм обробки помилок та винятків. У разі виявлення некоректного формату даних або відсутності очікуваних структур система активує спеціалізований обробник помилок. Цей компонент забезпечує генерацію інформативних повідомлень про характер несправності та реалізує стратегії відновлення для забезпечення стабільності роботи платформи. Система логує деталі помилок для подальшого аналізу та оптимізації процесів взаємодії з API.

Механізм обробки помилок включає диференційований підхід до різних типів несправностей. Система розрізняє тимчасові мережеві несправності, помилки формату даних та критичні збої серверної частини. Для кожного типу помилок реалізовані специфічні стратегії відновлення, включаючи автоматичні повторні спроби, використання кешованих даних та відображення інформативних повідомлень користувачу.

Фінальний етап виконання методу надає завершення операції та підготовку системи до обробки наступних користувацьких взаємодій. Система здійснює очищення тимчасових ресурсів, оновлення станів компонентів та забезпечення готовності до подальших операцій з каталогом продуктів. Цей процес включає оптимізацію використання пам'яті та налаштування кешування для покращення продуктивності майбутніх запитів.

Технічна реалізація методу базується на принципах модульної архітектури та розділення відповідальностей. Компонент інтегрується з іншими елементами клієнтської системи через чітко визначені інтерфейси та підтримує розширювані механізми конфігурації. Метод забезпечує масштабованість для роботи з великими каталогами продуктів та адаптивність до змін у структурі API серверної частини [17].

Система реалізує прогресивні стратегії завантаження даних, включаючи пагінацію та lazy loading для оптимізації продуктивності при роботі з великими наборами продуктів. Ці механізми забезпечують швидке початкове завантаження інтерфейсу та поступове отримання додаткових даних відповідно до потреб користувача, що дуже покращує загальний досвід взаємодії з платформою.

2.3 Розробка методу обробки навігації

Метод обробки навігації є центральним компонентом архітектури клієнтської частини e-commerce платформи для комп'ютерних ігор, що забезпечує динамічне управління переходами між різними розділами додатку та оптимізує користувацький досвід взаємодії з інтерфейсом. Цей метод реалізує комплексну систему маршрутизації, яка адаптується до контексту користувацьких дій та забезпечує плавні переходи між компонентами платформи (рис. 2.3).

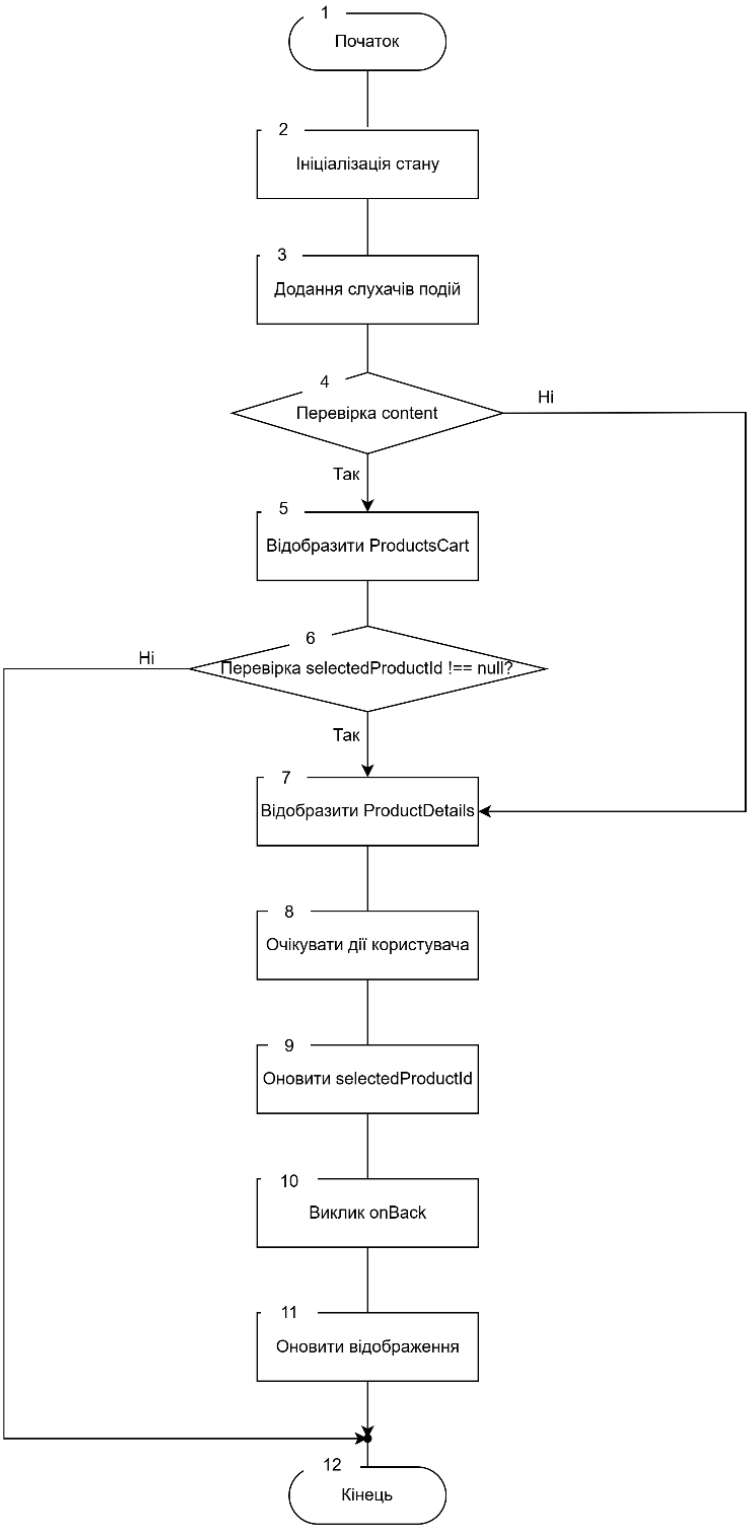

Рисунок 2.3 – Блок-схема методу обробки навігації

Початковий етап виконання методу навігації характеризується ініціалізацією базового стану системи маршрутизації. Система встановлює початкові параметри навігаційного контексту та підготовлює необхідні структури даних для подальшої обробки користувацьких переходів. Цей процес

включає налаштування слухачів подій, ініціалізацію історії навігації та конфігурацію базових параметрів маршрутизації, що забезпечують стабільну основу для функціонування всієї навігаційної системи.

Другим критично важливим етапом є процес ініціалізації стану навігаційних компонентів. Система здійснює налаштування початкових значень для всіх навігаційних елементів, включаючи активні маршрути, параметри переходів та контекстну інформацію про поточне розташування користувача в структурі додатку. Цей етап забезпечує коректну синхронізацію між різними компонентами інтерфейсу та гарантує консистентність навігаційного стану.

Третій етап надає додавання та конфігурацію слухачів подій для обробки користувацьких взаємодій з навігаційними елементами. Система реєструє обробники для різних типів навігаційних подій, включаючи кліки по посиланнях, зміни URL, використання кнопок браузера та програмні переходи. Ці слухачі забезпечують реактивність системи навігації та гарантують своєчасну обробку всіх користувацьких дій, пов'язаних з переміщенням по додатку.

Центральним елементом архітектури методу є система валідації контенту та визначення відповідних компонентів для відображення. Після отримання навігаційного запиту система аналізує тип контенту, який необхідно відобразити, та приймає рішення щодо вибору відповідного компонента. Цей процес включає перевірку доступності запитуваного контенту, валідацію прав доступу користувача та визначення оптимальної стратегії рендерингу для конкретного типу сторінки.

У випадку, коли система визначає необхідність відображення каталогу продуктів, активується спеціалізований компонент ProductsCart. Цей компонент забезпечує оптимізоване відображення списку ігрових продуктів з підтримкою різних режимів перегляду, фільтрації та сортування. Система здійснює налаштування параметрів компонента відповідно до контексту навігації та забезпечує дієве завантаження даних для відображення актуального каталогу продуктів [18].

Важливим аспектом навігаційної логіки є обробка переходів до детальних сторінок продуктів. Система включає механізм валідації ідентифікатора обраного продукту для визначення необхідності відображення детальної інформації. Процес валідації перевіряє наявність та коректність selectedProductId, що дозволяє системі прийняти обґрунтоване рішення щодо активації компонента ProductDetails або збереження поточного стану інтерфейсу.

При підтвердженні наявності валідного ідентифікатора продукту система активує компонент ProductDetails для відображення повної інформації про обраний ігровий продукт. Цей компонент забезпечує комплексне представлення всіх характеристик продукту, включаючи мультимедійний контент, технічні специфікації, ціни та опції покупки. Система оптимізує завантаження детальної інформації та забезпечує швидке відображення контенту користувачу.

Наступним етапом в архітектурі методу є очікування дій користувача після відображення детальної інформації про продукт. Система переходить в режим моніторингу користувацьких взаємодій та готується до обробки подальших навігаційних команд. Цей стан характеризується активним прослуховуванням подій інтерфейсу та підтримкою готовності до швидкої реакції на користувацькі дії.

Критично важливим компонентом методу є механізм оновлення selectedProductId при зміні контексту перегляду. Система забезпечує синхронізацію ідентифікатора обраного продукту з поточним станом інтерфейсу, що гарантує коректну роботу всіх пов'язаних компонентів та запобігає конфліктам в навігаційній логіці. Процес оновлення включає валідацію нових значень та каскадне оновлення залежних елементів інтерфейсу.

Реалізація функціональності повернення до попередньої сторінки здійснюється через виклик методу onBack, який забезпечує коректне відновлення попереднього стану навігації. Система підтримує історію переходів та забезпечує можливість дієвого повернення до попередніх розділів додатку з

збереженням контексту користувацької сесії. Цей механізм включає оптимізацію для швидкого відновлення стану та мінімізації повторних завантажень даних.

Завершальним етапом циклу навігації є оновлення відображення інтерфейсу відповідно до нового навігаційного стану. Система здійснює рендеринг оновлених компонентів, синхронізацію візуальних елементів та забезпечення консистентності всього користувацького інтерфейсу. Процес включає оптимізацію продуктивності рендерингу та мінімізацію візуальних артефактів під час переходів.

Архітектурно метод навігації реалізований з використанням принципів реактивного програмування та забезпечує високий рівень модульності. Система підтримує розширювані механізми конфігурації маршрутів, динамічне додавання нових навігаційних сценаріїв та інтеграцію з зовнішніми системами аналітики для моніторингу користувацької поведінки. Метод забезпечує оптимальну продуктивність через lazy loading компонентів та дієве управління пам'яттю під час навігаційних переходів.

2.4 Висновки

Розроблені методи клієнтської архітектури e-commerce платформи для комп'ютерних ігор демонструють комплексний підхід до створення дієвої системи взаємодії користувача з ігровим контентом. Аналіз трьох ключових методів виявив фундаментальні принципи побудови масштабованої та надійної клієнтської частини платформи.

Метод виводу деталей про продукт забезпечує високий рівень користувацького досвіду через реалізацію асинхронної взаємодії з серверними API та комплексну систему обробки помилок. Архітектурне рішення щодо використання graceful degradation дозволяє платформі зберігати функціональність навіть при тимчасових технічних збоях, що є критично важливим для комерційної системи. Впровадження механізмів кешування оптимізує продуктивність та зменшує навантаження на серверну інфраструктуру.

Метод виводу списків всіх продуктів демонструє дієвий підхід до роботи з великими обсягами даних через реалізацію прогресивних стратегій завантаження. Система валідації даних забезпечує стабільність роботи платформи при різних форматах API-відповідей, а диференційований механізм обробки помилок дозволяє системі адаптуватися до різних типів збоїв. Архітектурне рішення щодо підтримки пагінації та lazy loading забезпечує оптимальну продуктивність при роботі з каталогами великого розміру.

Метод обробки навігації реалізує складну логіку маршрутизації через інтеграцію реактивного програмування та динамічного управління компонентами. Система забезпечує плавні переходи між різними розділами платформи при мінімізації повторних завантажень даних. Архітектурне рішення щодо централізованого управління навігаційним станом гарантує консистентність користувацького інтерфейсу та дієву синхронізацію між компонентами.

Спільним аспектом всіх розроблених методів є впровадження принципів модульної архітектури та розділення відповідальностей. Це забезпечує високий рівень масштабованості системи та спрощує процеси подальшого розвитку платформи. Використання асинхронних патернів взаємодії в усіх методах гарантує збереження відзивчості користувацького інтерфейсу навіть при обробці складних операцій.

3 РОЗРОБКА МОДУЛІВ ПРОГРАМНОГО ЗАСОБУ

3.1 Розробка алгоритмів і програм e-commerce платформи для комп’ютерних ігор

Аналіз вхідних даних для програмної системи e-commerce платформи для комп'ютерних ігор потребує визначення основних інформаційних потоків, які формуються в процесі придбання, завантаження та використання ігрового контенту. Дані, що оброблятимуться системою, характеризуються різнорідністю, специфічною структурою та вимогами до обробки, що зумовлює необхідність комплексного підходу до їх організації та зберігання.

Для технічної реалізації системи обрано JavaScript у поєднанні з фреймворком React для клієнтської частини веб-інтерфейсу та Electron для створення крос-платформного десктопного додатку [19]. Архітектурна організація системи надає багаторівневу структуру з чітким розмежуванням відповідальності між компонентами та модульною організацією кодової бази. Такий підхід надасть гнучкості процесу розробки через можливість паралельної роботи над різними модулями та спрощуватиме подальшу підтримку системи через розділення і групування функціональності в окремих частинах системи.

Процес розробки e-commerce платформи для комп'ютерних ігор надає створення чотирьох основних функціональних модулів клієнтської архітектури: каталог ігрових продуктів, користувацький профіль, система платежів та модуль завантаження і управління ігровим контентом. Кожен із цих модулів оперує специфічними наборами даних, що потребують відповідних підходів до організації та відображення.

Модуль каталогу ігрових продуктів надаватиме функціональність для роботи з різнотипною інформацією, що охоплює не лише параметричні характеристики ігор (жанр, видавець, системні вимоги), але й користувацькі рейтинги, відгуки та мультимедійні дані. Основним призначенням цього модуля є дієве відображення та фільтрація як чітко структурованої параметричної інформації, так і пов'язаних зображень, відео-трейлерів та демонстраційних

матеріалів, які зберігатимуться на віддаленому сервері та завантажуватимуться за допомогою REST API .

Модуль користувацького профілю опрацьовуватиме дві основні категорії даних, які відрізняються за структурою та характером формування: особисті дані користувача та інформація про ігрову бібліотеку. Особисті дані включатимуть обов'язкові для автентифікації поля (логін, електронна пошта, хешований пароль) та опціональну інформацію (ім'я, контактні дані, аватар). Ігрова бібліотека міститиме дані про придбані ігри, їхній стан завантаження, час гри та досягнення. Для дієвої роботи з такими даними оптимальним вибором є використання локального сховища (LocalStorage, IndexedDB) для кешування даних профілю та збереження налаштувань користувача, а також Redux для управління станом додатку в реальному часі [20].

Модуль системи платежів відповідатиме за інтеграцію з платіжними шлюзами та забезпечення безпечного проведення транзакцій. Призначення цієї частини системи полягає в оперуванні метаданими платежів, такими як тип платежу, сума, дата операції, статус обробки sta відомості про покупку. Сам процес проведення платежів буде реалізовано через використання сторонніх платіжних API (Stripe, PayPal), що забезпечить необхідний рівень безпеки даних. Для цього модуля важливо забезпечити надійне збереження історії транзакцій та можливість відстеження їхнього статусу.

Модуль завантаження і управління ігровим контентом оперуватиме даними про процеси завантаження ігор, їх встановлення та оновлення. Цей модуль забезпечуватиме управління інформацією про доступні оновлення, прогрес завантаження, статус встановлення та цілісність файлів. Для дієвого функціонування системи критично важливим є забезпечення можливості відновлення завантаження після збоїв та паралельної обробки декількох завантажень одночасно. Для реалізації цього модуля доцільно використовувати можливості Electron для роботи з файловою системою та Node.js для управління процесами завантаження [21].

У якості засобів для комунікації з серверною частиною доцільно використати RESTful API для основних операцій з каталогом, профілем та платежами, а також WebSocket для обміну даними в реальному часі при завантаженні ігор та оновленні статусу користувацької бібліотеки. Для локального зберігання даних підійде IndexedDB, яка дозволить дієво кешувати об'ємну інформацію про ігри та забезпечить швидкий доступ до неї навіть без підключення до мережі.

Отже, аналіз вхідних даних системи e-commerce платформи для комп'ютерних ігор визначає необхідність розробки чотирьох функціональних модулів, кожен з яких працює зі специфічними наборами даних та вимагає відповідних підходів до їх відображення та обробки. Реалізація системи на основі React для веб-інтерфейсу та Electron для десктопного додатку забезпечить надійну основу для розробки багаторівневої архітектури. Комбінований підхід до організації даних з використанням серверного API для отримання основної інформації та локальних сховищ для кешування і управління станом дозволить досягти оптимального балансу між продуктивністю роботи та зручністю використання. Використовуючи такий набір технологій при ретельному плануванні та проєктуванні можна створити програмний продукт, який дозволить оптимізувати процес придбання та використання комп'ютерних ігор.

Розробка серверної архітектури e-commerce платформи для комп'ютерних ігор потребує чіткого розуміння основних функціональних компонентів та послідовності їх взаємодії. Модель системи зображає собою послідовний потік дій користувача від моменту входу до завершення процесу придбання гри [22].

Процес починається з точки входу користувача на сайт платформи. На цьому етапі система визначає статус відвідувача – новий чи існуючий користувач. Для нових користувачів передбачено процедуру реєстрації, де збираються необхідні персональні дані, створюється обліковий запис та налаштовуються параметри безпеки. Існуючим користувачам пропонується процедура авторизації із введенням облікових даних та можливістю відновлення доступу при необхідності.

Після успішної автентифікації користувач потрапляє на головну сторінку платформи, що слугує центральним вузлом навігації. Звідси користувач може обрати один із трьох основних напрямків: перегляд повного каталогу ігор, застосування фільтрації за категоріями для пошуку конкретних ігор, або перехід до особистого кабінету для управління власними даними та перегляду історії активності (рис. 3.1).

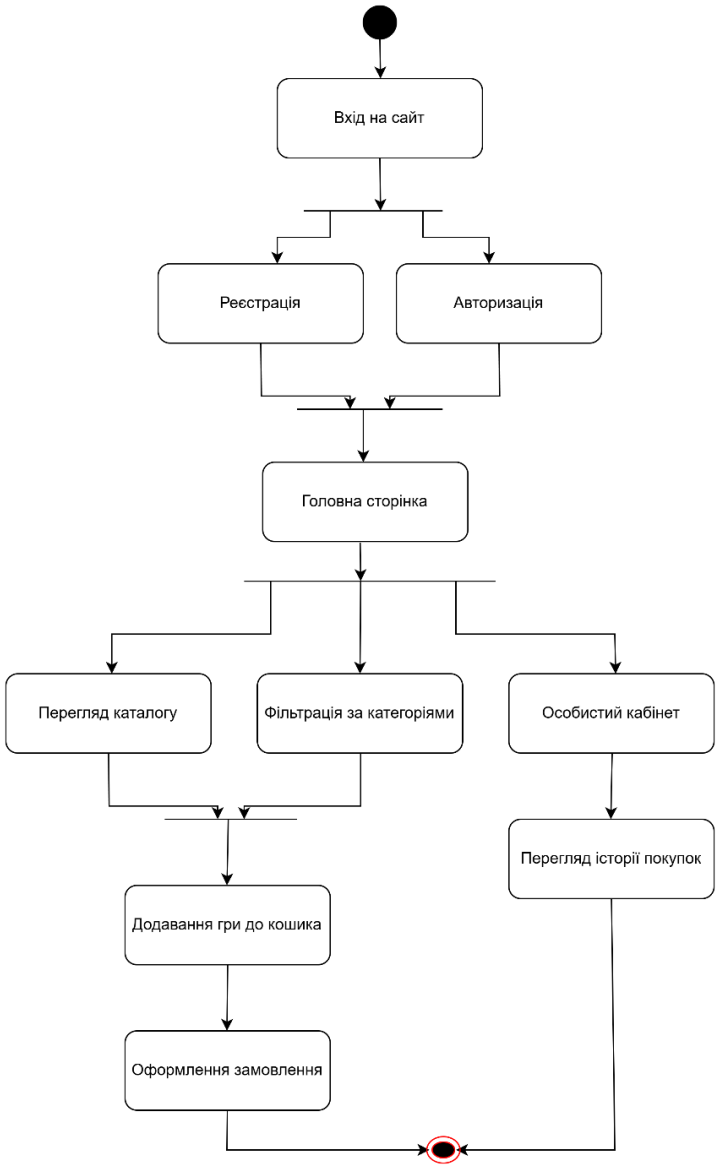

Рисунок 3.1 – Діаграма діяльності програмних засобів

При перегляді каталогу користувачу доступний повний асортимент ігрових продуктів з детальними описами, скріншотами, відгуками та

рейтингами. Функціонал фільтрації дозволяє звузити пошук за жанрами, видавцями, ціновим діапазоном, датою випуску та іншими релевантними параметрами. Обидва ці шляхи ведуть до можливості додавання обраної гри в кошик для подальшого придбання.

Особистий кабінет надає доступ до історії попередніх покупок, дозволяючи відстежувати придбані ігри, завантажувати їх повторно, керувати підписками та переглядати трансакції. Вся ця функціональність забезпечує користувачеві повний контроль над власним ігровим портфоліо.

Після додавання гри в кошик користувач переходить до процесу оформлення замовлення. На цьому етапі відбувається вибір способу оплати, застосування промокодів чи бонусів, підтвердження деталей замовлення та завершення процесу покупки. Система обробляє платіжну інформацію, генерує ключі активації або надає посилання на завантаження цифрового контенту.

Процес завершується успішною покупкою, після чого користувач отримує підтвердження про придбання та доступ до придбаного контенту. Архітектурно ця модель реалізується через взаємодію серверних модулів автентифікації, управління каталогом, обробки кошика, платіжного шлюзу та системи доставки цифрового контенту, що забезпечує повний цикл обслуговування клієнта від входу до отримання продукту.

3.2 Розробка модуля отримання продуктів з серверної частини

Важливим компонентом e-commerce платформи для комп'ютерних ігор є модуль взаємодії з серверною частиною, який відповідає за отримання даних про продукти та інформацію про користувача. Даний модуль забезпечує ключовий функціонал для відображення каталогу ігор, формування кошика покупок та персоналізації користувацького досвіду.

Реалізація модуля отримання даних з серверної частини базується на використанні React Hooks, зокрема хуку useEffect, що дозволяє виконувати побічні ефекти в функціональних компонентах. Це архітектурне рішення

відповідає сучасним тенденціям розробки React-додатків та забезпечує дієве керування життєвим циклом компонентів.

Основою модуля є функціональна логіка, яка виконується при монтуванні компонента завдяки хуку useEffect з порожнім масивом залежностей. Такий підхід гарантує, що запити до API будуть виконані лише один раз при початковому завантаженні компонента, що запобігає надмірним мережевим запитам та оптимізує продуктивність додатку (рис. 3.2).

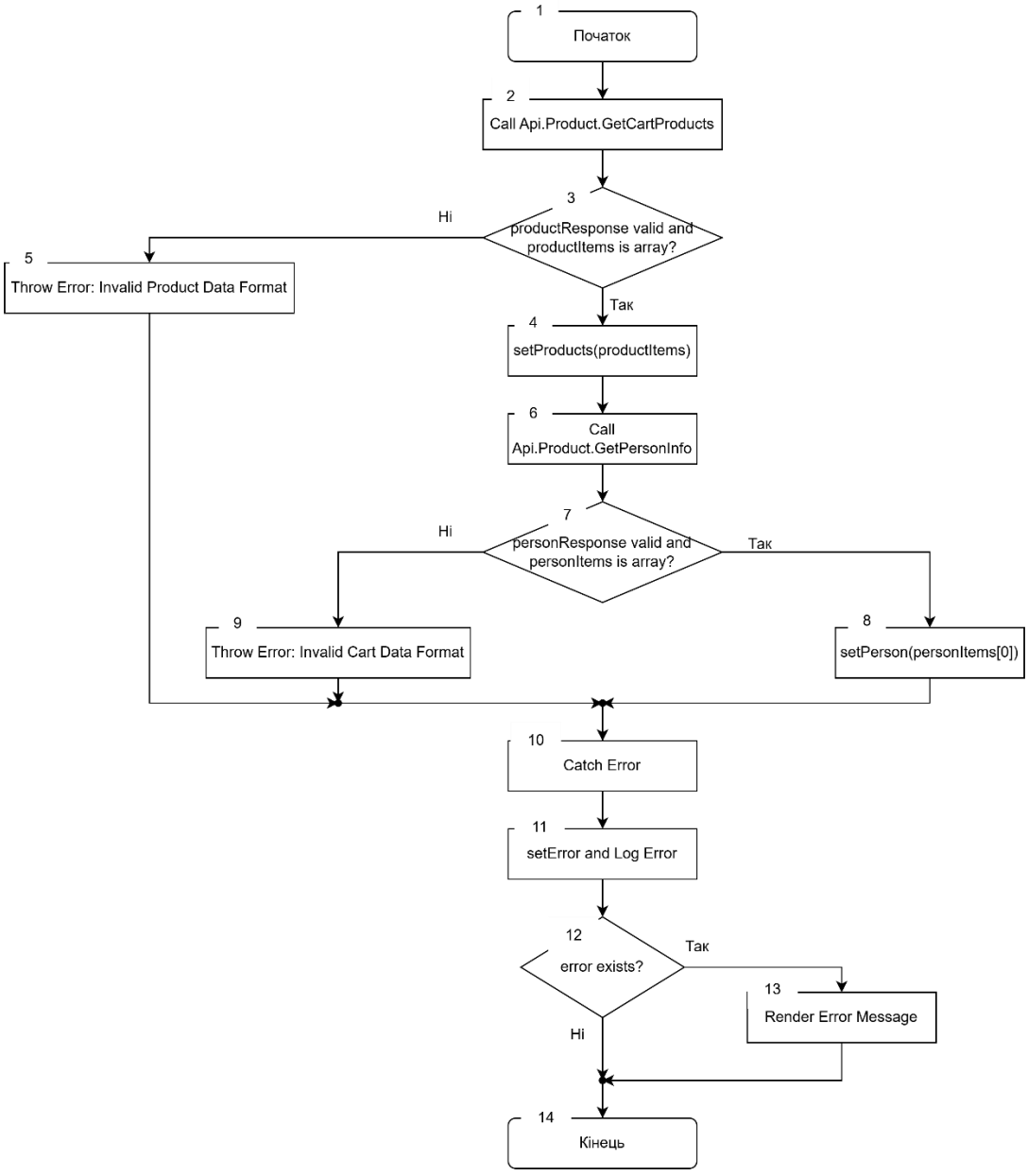

Рисунок 3.2 – Блок схема модуля отримання продуктів з серверної частини

Для взаємодії з API використовується абстракція у вигляді фабричного методу Api(), який повертає об'єкт з доступом до різних ендпоінтів серверної частини. Така архітектура дозволяє ізолювати логіку взаємодії з сервером від компонентів користувацького інтерфейсу, що покращує тестованість коду та спрощує його підтримку.

Модуль здійснює два основних запити: отримання товарів у кошику через метод GetCartProducts() та отримання даних користувача через метод GetPersonInfo(). Кожен запит супроводжується валідацією отриманих даних, що є критично важливим для забезпечення надійності додатку. Валідація перевіряє наявність відповіді від сервера та коректність структури даних, зокрема наявність масивів productItems та personItems.

Особливої уваги заслуговує обробка помилок, реалізована за допомогою конструкції try-catch. Цей механізм дозволяє елегантно обробляти як помилки мережевого з'єднання, так і несправності з форматом даних. У випадку виникнення виключної ситуації, код встановлює значення стану error через виклик функції setError, що призводить до відображення повідомлення про помилку користувачеві. Додатково деталі помилки логуються в консолі для полегшення діагностики несправностей під час розробки та тестування.

Після успішного отримання даних, вони зберігаються у відповідних станах компонента за допомогою функцій setProducts та setPerson. Ці функції є частиною хуків стану (useState), що забезпечує реактивне оновлення інтерфейсу при зміні даних.

Варто відзначити, що поточна реалізація модуля надає синхронні виклики API, що може бути неоптимальним у випадку повільного з'єднання або великого обсягу даних. У майбутніх версіях планується модифікація коду для використання асинхронних запитів з async/await синтаксисом, що дозволить покращити відгук інтерфейсу під час завантаження даних.

Додаткової уваги потребує обробка стану завантаження. В поточній реалізації відсутній індикатор прогресу, що може викликати незручності при

повільному з'єднанні. Рекомендується розширити функціонал шляхом додавання стану isLoading та відповідного індикатора в інтерфейсі.

Рендеринг компонента включає перевірку наявності помилок і відображення відповідного повідомлення у випадку їх виникнення.

Такий підхід до обробки помилок забезпечує інформативний зворотний зв'язок для користувача, що покращує загальний досвід використання платформи навіть у випадку технічних несправностей.

Розроблений модуль отримання продуктів з серверної частини є важливим компонентом клієнтської архітектури e-commerce платформи для комп'ютерних ігор. Він забезпечує надійне отримання та валідацію даних, що є фундаментом для коректної роботи інших модулів системи, таких як каталог ігрових продуктів, кошик покупок та профіль користувача. Використання сучасних підходів до розробки React-додатків забезпечує дієве функціонування модуля та його інтеграцію в загальну архітектуру системи.

3.3 Розробка модуля маршрутизації сторінок

Модуль маршрутизації сторінок є критичним компонентом клієнтської архітектури e-commerce платформи для комп'ютерних ігор, оскільки забезпечує навігацію між різними інтерфейсами системи. Цей модуль визначає логіку переходів між сторінками та управляє відображенням відповідних компонентів залежно від дій користувача.

Реалізація маршрутизації в даному проєкті використовує підхід, заснований на керуванні станом компонента, а не традиційну бібліотеку маршрутизації, таку як React Router. Це рішення дозволяє забезпечити простий та легкий у розумінні механізм навігації для додатку, що має обмежену кількість основнихзображень [23].

Основою модуля маршрутизації є компонент App, який виступає контейнером для всіх інших компонентів системи. Компонент використовує React Hooks для керування станом, зокрема два основних стани: selectedProductId та content (рис. 3.3).

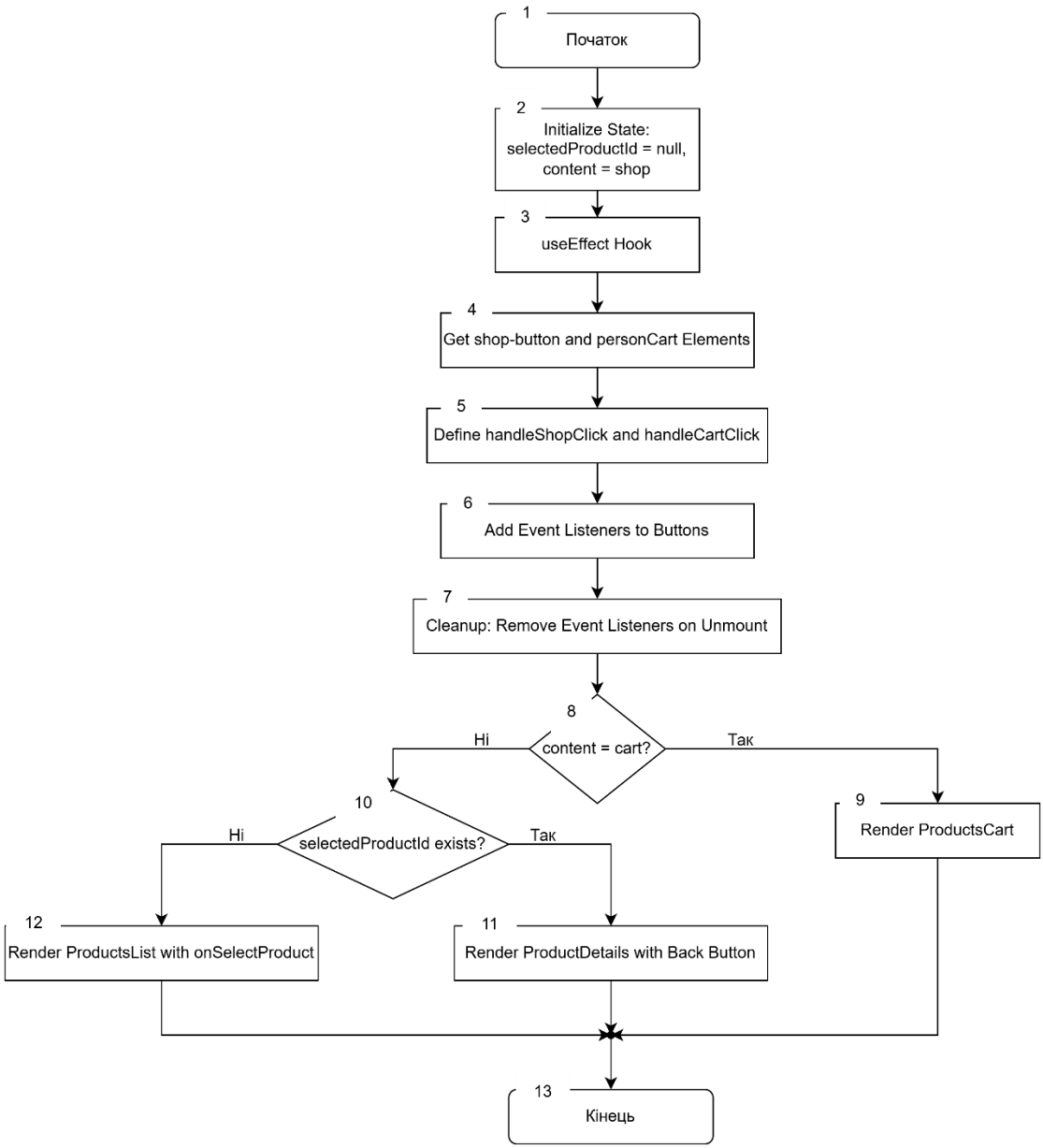

Рисунок 3.3 – Блок схема модуля маршрутизації сторінок

Стан selectedProductId відповідає за відстеження ідентифікатора обраного користувачем продукту. Коли цей стан встановлено в null, система відображає загальний список продуктів. Коли стан містить конкретний ідентифікатор, система переходить до детального перегляду відповідного продукту. Цей підхід дозволяє реалізувати навігацію між списком продуктів та деталями конкретного продукту.

Стан content визначає основний режим інтерфейсу, який може бути встановлений у значення «shop» (відображення магазину) або «cart»

(відображення кошика покупок). Це дозволяє організувати переключення між двома основними інтерфейсами додатку.

Для обробки подій навігації компонент використовує хук useEffect, який встановлює обробники подій натискання на елементи інтерфейсу, що відповідають за навігацію.

Цей хук демонструє декілька важливих підходів до реалізації маршрутизації:

- встановлення обробників подій для кнопок навігації, які ідентифікуються за допомогою їх ідентифікаторів DOM («shop-button» та «personCart»);
- визначення функцій-обробників подій, які змінюють стан додатку при натисканні на відповідні кнопки;
- очищення обробників подій при демонтажі компонента, що є важливим для запобігання витокам пам'яті.

Функція handleShopClick встановлює режим відображення магазину та скидає вибір конкретного продукту, що призводить до показу загального списку продуктів. Функція handleCartClick аналогічно встановлює режим відображення кошика та також скидає вибір продукту.

Важливо відзначити використання порожнього масиву залежностей у хуку useEffect, що гарантує виконання цього ефекту лише один раз при монтуванні компонента. Це запобігає зайвим повторним встановленням обробників подій.

Рендерінг відповідного інтерфейсу базується на умовних операторах, які аналізують поточний стан компонента.

Ця частина використовує тернарні оператори для вибору, який компонент відображати:

- якщо content встановлено в «cart», відображається компонент ProductsCart, що зображає кошик покупок;
- якщо вибрано конкретний продукт (selectedProductId не є null), відображається компонент ProductDetails з передачею ідентифікатора продукту та функції для повернення до списку продуктів;

- в інших випадках відображається компонент ProductsList, який показує загальний каталог продуктів.

Особливу увагу слід звернути на передачу пропсів між компонентами. Компоненту ProductDetails передається не тільки ідентифікатор продукту, але й функція onBack, яка дозволяє користувачу повернутися до загального списку продуктів. Це реалізується шляхом встановлення selectedProductId в null через стрілкову функцію () => setSelectedProductId(null).

Аналогічно, компоненту ProductsList передається функція onSelectProduct, яка дозволяє йому повідомити батьківський компонент про вибір конкретного продукту користувачем. Ця функція змінює стан selectedProductId, що призводить до переходу на сторінку деталей продукту.

Розроблений модуль маршрутизації сторінок забезпечує базову навігацію між основними зображеннями e-commerce платформи для комп'ютерних ігор. Його архітектура дозволяє користувачам легко переміщатися між списком продуктів, детальним переглядом конкретного продукту та кошиком покупок. Проста, але дієва реалізація на основі стану компонента забезпечує достатню функціональність для поточних потреб системи, залишаючи можливість для подальшого масштабування та покращення.

3.4 Розробка модуля додавання до кошика товару

Важливим функціональним елементом e-commerce платформи для комп'ютерних ігор є модуль додавання товарів до кошика, який забезпечує можливість формування замовлення користувачем. Цей модуль реалізує взаємодію між клієнтською та серверною частиною системи, виконуючи асинхронні запити для оновлення стану кошика покупок.

Реалізація модуля додавання до кошика базується на асинхронній функції handleAddToCart, яка викликається при взаємодії користувача з відповідним елементом інтерфейсу, найчастіше - кнопкою «Додати до кошика». Ця функція відповідає за виконання HTTP-запиту до серверного API та обробку результату цього запиту (рис. 3.4).

У наведеному фрагменті коду функція використовує конструкцію async/await для виконання асинхронних операцій, що дозволяє писати асинхронний код у синхронному стилі, покращуючи його читабельність та спрощуючи обробку помилок. Цей підхід є сучасною практикою в JavaScript-розробці та забезпечує більш чистий та зрозумілий код порівняно з використанням callbacks або чистих Promise.

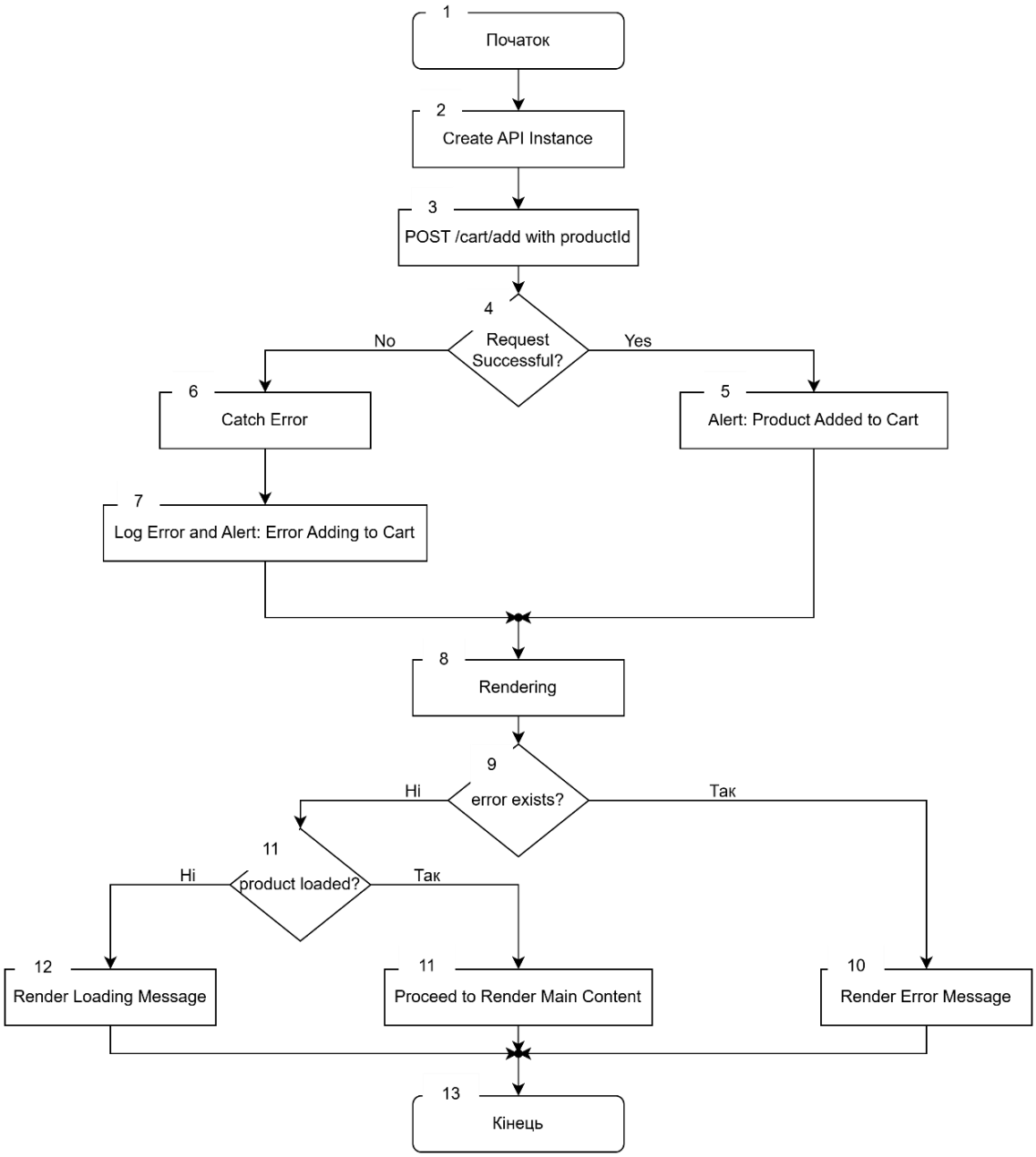

Рисунок 3.4 – Реалізація функції handleAddToCart

Всередині функції відбувається створення екземпляра API-клієнта через виклик фабричного методу Api(). Цей клієнт надає інтерфейс для взаємодії з різними ендпоінтами серверного API. В даному випадку використовується метод post для відправлення POST-запиту на ендпоінт '/cart/add' з передачею ідентифікатора продукту (productId) у тілі запиту. Такий підхід відповідає принципам REST API, де POST-запити використовуються для створення нових ресурсів або модифікації існуючих.

Особливу увагу у розробці модуля було приділено обробці помилок, яка реалізована за допомогою конструкції try-catch. У випадку успішного виконання запиту користувачу відображається повідомлення про успішне додавання товару до кошика за допомогою функції alert. Хоча використання alert є простим рішенням для відображення повідомлень, у продакшн-версії додатку рекомендується замінити його на більш сучасний та гнучкий компонент сповіщень, який краще інтегрується в загальний дизайн інтерфейсу.

У випадку виникнення помилки під час виконання запиту, модуль забезпечує логування деталей помилки в консоль за допомогою console.error та відображення користувачу загального повідомлення про помилку. Такий підхід дозволяє розробникам мати доступ до технічних деталей помилки для її діагностики, при цьому не перевантажуючи користувача технічною інформацією.

Для покращення користувацького досвіду, модуль також включає обробку станів завантаження та помилок при рендерингу інтерфейсу.

Особливістю розробленого модуля є його інтеграція в загальну архітектуру додатку через використання спільного API-клієнта, що забезпечує єдиний підхід до комунікації з серверною частиною. Такий підхід спрощує підтримку та масштабування системи, оскільки будь-які зміни в механізмі комунікації можуть бути впроваджені централізовано.

Реалізований модуль додавання до кошика товару забезпечує критично важливу функціональність e-commerce платформи для комп'ютерних ігор, дозволяючи користувачам формувати замовлення через інтуїтивно зрозумілий

інтерфейс. Асинхронна взаємодія з серверною частиною системи, обробка помилок та інформативний зворотний зв'язок для користувача створюють основу для позитивного користувацького досвіду при здійсненні покупок на платформі.

3.5 Розробка модуля інтерфейсу користувача

Проектування інтерфейсу e-commerce платформи для комп'ютерних ігор вимагає особливого підходу, зважаючи на специфіку взаємодії користувачів з ігровим контентом та потреби цільової аудиторії. Розроблений інтерфейс забезпечує швидкий доступ до ключових функцій платформи через інтуїтивно зрозумілу навігацію та візуально привабливе оформлення, що відповідає сучасним тенденціям дизайну ігрових платформ.

Основою візуальної концепції інтерфейсу стало темне кольорове оформлення, обране з урахуванням особливостей використання платформи геймерською аудиторією. Темна палітра знижує навантаження на зір під час тривалого використання додатку, особливо у вечірній та нічний час, що є типовим сценарієм для багатьох геймерів. Також такий дизайн дозволяє дієвіше підкреслити яскраві графічні елементи ігрових обкладинок та промо-матеріалів, створюючи необхідний контраст.

Структурно інтерфейс побудовано за принципом асиметричного розподілу екранного простору з розміщенням вертикальної навігаційної панелі у лівій частині екрану. Така організація забезпечує зручний доступ до основних розділів платформи незалежно від того, на якій сторінці перебуває користувач. Навігаційне меню включає критично важливі розділи: «Магазин», «Сортування» та «Особистий кабінет», що охоплюють весь базовий функціонал платформи. Для підвищення інтуїтивності інтерфейсу кожен пункт меню супроводжується відповідною іконографікою, що полегшує візуальну ідентифікацію розділів.

Центральна частина інтерфейсу зображено у вигляді адаптивної контентної області, яка динамічно змінюється залежно від вибраного розділу чи функції. Реалізація цього компоненту здійснена за допомогою React Router, що дозволяє оновлювати контент без перезавантаження всього додатку,

забезпечуючи плавні переходи між сторінками та покращуючи загальний користувацький досвід.

Сторінка профілю користувача реалізована як персоналізований інформаційний хаб, де користувач бачить своє ім'я (наприклад, «Vorsinok») та колекцію придбаних ігор. Кожна гра у бібліотеці зображена візуальною карткою із обкладинкою, назвою та статусом доступності. Для ігор, доступних до встановлення, реалізовано інтерактивну кнопку «Завантажити», що активує відповідний процес через модуль завантаження контенту. Дизайн цієї сторінки фокусується на швидкому доступі до вже придбаних ігор та мінімізації кроків, необхідних для початку гри (рис. 3.5).

Рисунок 3.5 – Особистий кабінет користувача

Реалізація сторінки опису гри втілює концепцію інформаційної повноти при збереженні візуальної чистоти. На прикладі гри «Black Myth: Wukong» можна бачити детальний інформаційний блок, який містить високоякісні зображення, текстовий опис та технічні характеристики (рис. 3.6). Особливу увагу приділено структуруванню даних про системні вимоги, які розділені на мінімальні та рекомендовані, з чітким відображенням вимог до операційної системи, процесора, оперативної пам'яті, відеокарти та дискового простору. В нижній частині розташована кнопка «До кошика» з чітко вказаною ціною (2.1 EUR), розроблена таким чином, щоб спонукати до дії при збереженні інформативності.

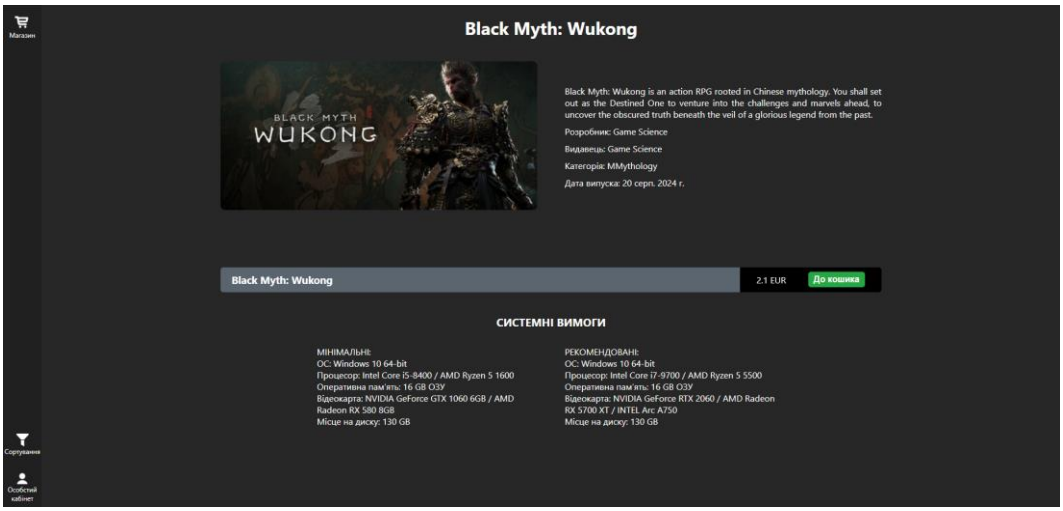

Рисунок 3.6 – Сторінка гри

Каталог ігор реалізовано як масив карток із візуально привабливим розташуванням. Кожна картка функціонує як самодостатній інформаційний елемент, що містить обкладинку гри, її назву, ціну та лаконічний опис із датою релізу. Цей підхід дозволяє користувачеві швидко сканувати доступні пропозиції та приймати обґрунтовані рішення щодо переходу до детальної інформації про конкретну гру. Розміщення карток оптимізовано для різних розмірів екрану з автоматичним перебудуванням сітки при зміні роздільної здатності чи орієнтації пристрою (рис. 3.7).

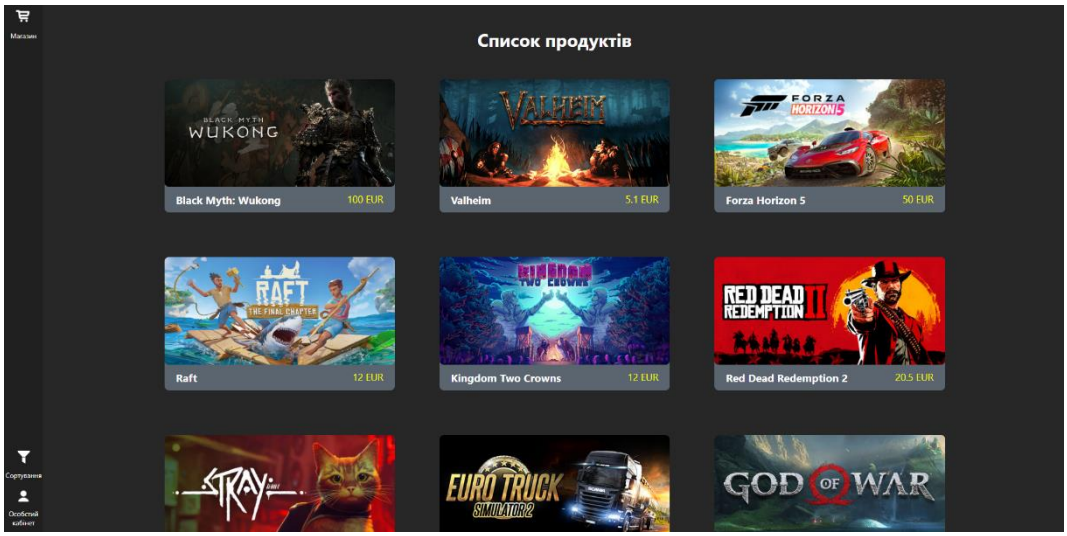

Рисунок 3.7 – Список продуктів

Технічна реалізація інтерфейсу базується на компонентному підході React, що забезпечує модульність і перевикористання елементів UI. Для стилізації використано сучасні інструменти, такі як CSS модулі з можливістю ізоляції стилів для окремих компонентів, що запобігає конфліктам стилів при масштабуванні проєкту. Інтеграція з Electron дозволяє використовувати веб-технології для створення нативного настільного досвіду з повним доступом до системних API, необхідних для завантаження та встановлення ігрового контенту.

Важливим аспектом розробленого інтерфейсу є його відповідність принципам UX/UI-дружності, що проявляється у логічному розміщенні елементів, уніфікованій системі сповіщень та зрозумілій візуальній ієрархії. Модульна архітектура забезпечує незалежну розробку та тестування окремих компонентів, таких як профіль користувача, картки ігор, детальні описи та функціонал кошика. Впроваджена адаптивність дозволяє комфортно працювати з додатком на пристроях з різними характеристиками дисплеїв, забезпечуючи консистентний досвід користування незалежно від розміру вікна.

У результаті розробки інтерфейсу створено цілісну систему взаємодії користувача з e-commerce платформою для комп'ютерних ігор, яка поєднує естетичну привабливість із функціональною дієвістю, забезпечуючи зручний доступ до всіх ключових операцій: перегляду каталогу, ознайомлення з детальною інформацією, здійснення покупки та управління ігровою бібліотекою.

3.6 Висновки

Отже розроблені модулі отримання даних з серверної частини, маршрутизації сторінок та додавання товарів до кошика створюють надійну основу для функціонування системи, забезпечуючи стабільну взаємодію між клієнтською та серверною частинами через RESTful API.

Створений інтерфейс користувача відображає сучасні тенденції в дизайні ігрових платформ, поєднуючи темне кольорове оформлення з інтуїтивною навігацією та адаптивним розташуванням елементів.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування розробленої e-commerce платформи для комп'ютерних ігор проводилося з використанням інструментів розробника браузера Chrome DevTools та спеціалізованих засобів аналізу веб-продуктивності. Основна увага приділялася перевірці функціональності клієнтської частини, продуктивості завантаження ресурсів та оптимізації користувацького досвіду [24].

Перший етап тестування включав перевірку базової функціональності інтерфейсу каталогу продуктів. Система демонструє коректне відображення ігрових продуктів з відповідною візуалізацією та ціновою інформацією. Інтерфейс підтримує українську локалізацію, що підтверджується заголовком «Список продуктів» та правильним відображенням текстових елементів. DevTools показує стабільну роботу основних компонентів без критичних помилок у консолі браузера (рис. 4.1).

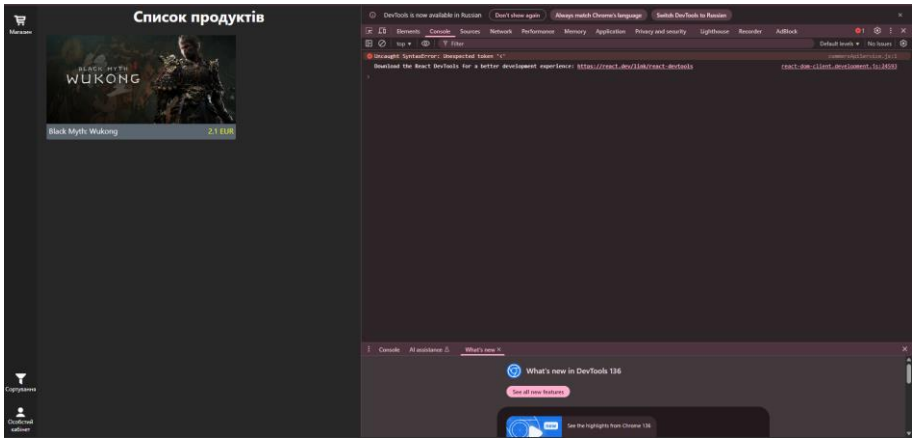

Рисунок 4.1 – Результат перевірки базової функціональності інтерфейсу

Другий етап тестування зосереджувався на аналізі продуктивності системи через вкладку Network у DevTools. Результати показують оптимальні показники мережевої активності з мінімальними значеннями основних метрик продуктивності. Largest Contentful Paint становить 0,23 секунди, що дуж нижче рекомендованого порогу в 2,5 секунди. Cumulative Layout Shift дорівнює 0,00,

що вказує на відсутність несподіваних зсувів елементів інтерфейсу під час завантаження. Interaction to Next Paint також показує нульове значення, що свідчить про миттєву реакцію на користувацькі взаємодії (рис. 4.2).

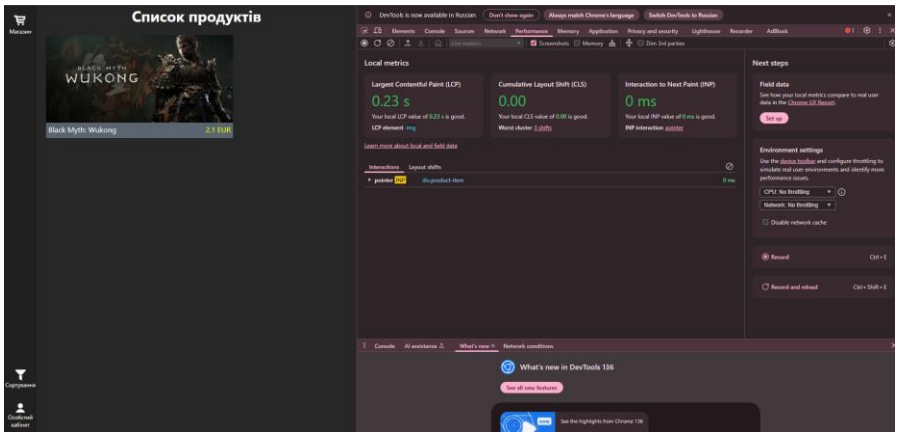

Рисунок 4.2 – Результат аналізу продуктивності системи

Детальний аналіз мережевого трафіку виявив дієву організацію завантаження ресурсів (рис. 4.3). Загальна кількість HTTP-запитів становить 11, що є оптимальним показником для сучасних веб-додатків. Загальний обсяг переданих даних складає 1,0 МБ, а фінішний час завантаження становить 1,13 секунди. Система завантажує різноманітні типи ресурсів, включаючи CSS стилі, PNG зображення, JavaScript файли та WebSocket з'єднання для динамічної взаємодії.

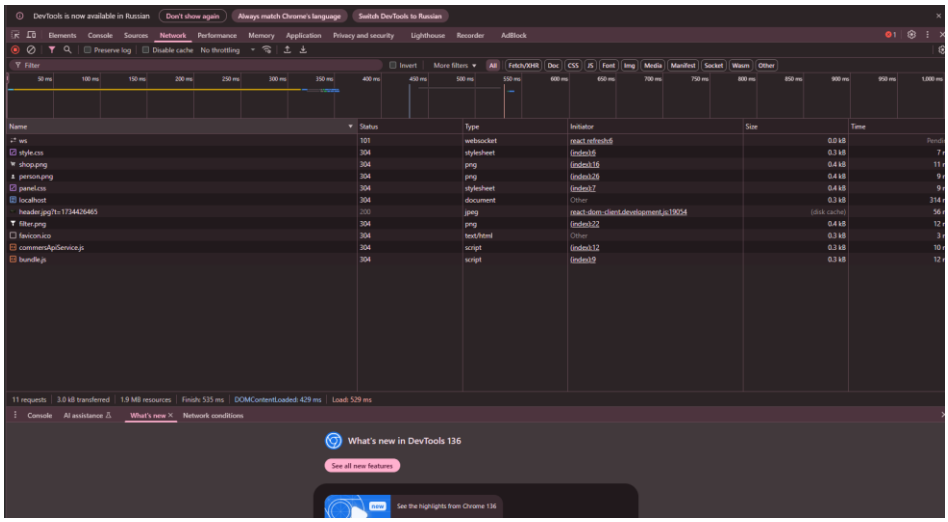

Рисунок 4.3 – Результат аналізу мережевого трафіку

Аналіз структури завантажених файлів показує правильну організацію фронтенд архітектури. Присутні основні компоненти React додатку, включаючи bundle.js для логіки додатку, style.css для стилізації, та різноманітні зображення продуктів. WebSocket з'єднання забезпечує real-time оновлення даних каталогу без необхідності перезавантаження сторінки.

Тестування SEO-оптимізації проводилося за допомогою розширення PR-CY для Chrome. Результати демонструють базовий рівень оптимізації з потенціалом для покращення. Мета-заголовок має довжину 9 символів з максимально допустимими 60, що вказує на необхідність його розширення для кращої індексації пошуковими системами (рис. 4.4).

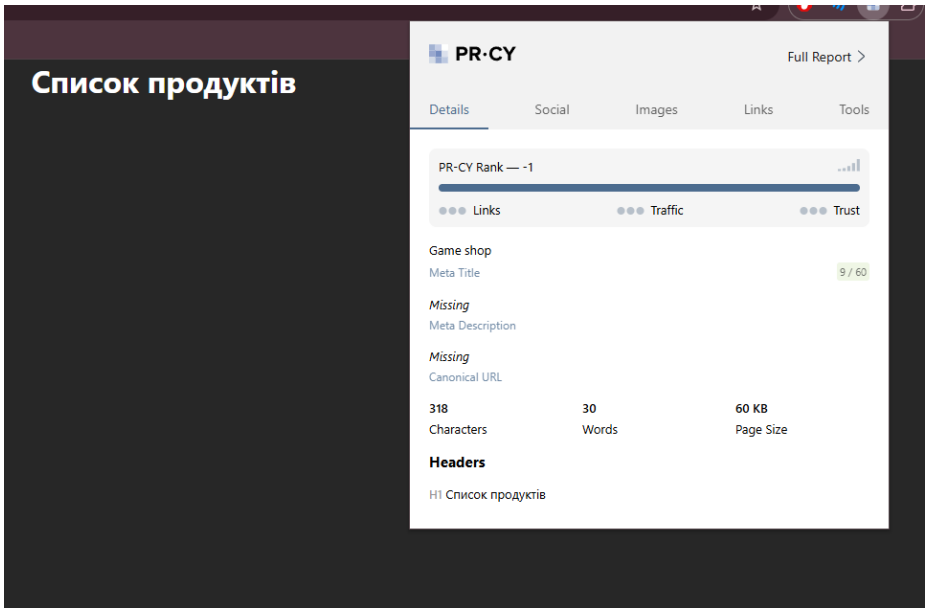

Рисунок 4.4 – Результат тестування SEO-оптимізації

Загальний обсяг сторінки становить 60 КБ при 318 символах тексту та 30 словах, що вказує на мінімалістичний підхід до дизайну з акцентом на візуальних елементах. Структура заголовків включає правильно оформлений H1 з текстом «Список продуктів», що забезпечує семантичну коректність HTML-розмітки.

Результати тестування підтверджують високу продуктивність та стабільність клієнтської частини e-commerce платформи. Система демонструє швидке завантаження, відсутність критичних помилок та дієве використання

мережевих ресурсів, що забезпечує позитивний користувацький досвід при роботі з каталогом ігрових продуктів.

4.2 Розробка інструкції користувача

Інструкція користувача надає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5/AMD Ryzen 5 або потужніший	Intel Core i3/AMD Ryzen 3
Оперативна пам'ять	ГБ RAM	4 ГБ RAM
Вільний простір на диску	1 ГБ	1 ГБ
Відеокарта	RX 580 або аналогічна	RX 580 або аналогічна
Операційна система	Windows 10 або macOS Big Sur	Windows 7 або macOS Mojave

Завантажте інсталяційний файл платформи з офіційного сайту відповідно до вашої операційної системи:

- Windows: EGameStore_Setup.exe;
- macOS: EGameStore.dmg;
- Linux: EGameStore.AppImage або EGameStore.deb.

Запустіть інсталяційний файл та слідуйте інструкціям майстра встановлення:

- оберіть мову інтерфейсу;
- прийміть умови ліцензійної угоди;
- виберіть директорію для встановлення;
- визначте додаткові параметри (створення ярлика на робочому столі, запуск при старті системи);

- натисніть кнопку «Встановити».

Після завершення встановлення запустіть програму через створений ярлик або з меню програм операційної системи.

Після першого запуску програмного забезпечення користувач потрапляє на екран вітання, який містить форму авторизації або реєстрації. Для початку роботи з платформою необхідно:

Створити обліковий запис:

- натисніть кнопку «Зареєструватися»;
- заповніть обов'язкові поля: електронна пошта, пароль, ім'я користувача;
- підтвердіть реєстрацію через отриманий лист на вказану електронну пошту.

Увійти в систему:

- введіть електронну пошту та пароль;
- натисніть кнопку «Увійти»;
- за бажанням можна встановити прапорець «Запам'ятати мене».

4.3 Висновки

Отже, було проведено тестування створених програмних модулів клієнтської архітектури e-commerce платформи для комп’ютерних ігор. Було досліджено дії системи при введенні коректних та некоректних даних. Також, було створено інструкцію користувача з експлуатації програмної системи.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено комплексне дослідження та розробку клієнтської архітектури e-commerce платформи для комп'ютерних ігор, що дозволило досягти поставлених цілей та отримати практично значущі результати.

Проведений порівняльний аналіз існуючих e-commerce платформ у сфері цифрової дистрибуції ігор виявив суттєві прогалини в існуючих рішеннях та підтвердив актуальність розробки інноваційної платформи. Дослідження показало, що жодна з аналізованих платформ не забезпечує оптимального поєднання всіх критично важливих аспектів сучасного ігрового ринку, що створює значні можливості для створення конкурентоспроможного продукту.

Науково обґрунтовано вибір технологічної платформи React/Electron, який забезпечує необхідну кросплатформність, гнучкість архітектури та високу швидкість розробки. Ця технологічна комбінація дозволяє дієво адаптуватися до динамічних змін ринку та забезпечує єдиний користувацький досвід взаємодії незалежно від типу пристрою.

Розроблено три ключові методи клієнтської архітектури, які формують надійну основу для функціонування e-commerce платформи:

Метод виводу деталей продукту реалізує асинхронну взаємодію з серверними API з комплексною системою обробки помилок та механізмами graceful degradation. Це забезпечує стабільність роботи системи навіть при тимчасових технічних збоях та оптимізує продуктивність через впровадження механізмів кешування.

Метод виводу списків продуктів демонструє дієвий підхід до роботи з великими обсягами даних через прогресивні стратегії завантаження, систему валідації даних та підтримку пагінації з lazy loading. Це забезпечує оптимальну продуктивність при роботі з великими каталогами ігор.

Метод обробки навігації реалізує складну логіку маршрутизації через інтеграцію реактивного програмування та динамічне управління компонентами, гарантуючи плавні переходи та консистентність користувацького інтерфейсу.

Всі розроблені методи об'єднані принципами модульної архітектури та розділення відповідальностей, що забезпечує високу масштабованість системи та спрощує процеси подальшого розвитку.

Створено повнофункціональні програмні модули для отримання даних з серверної частини, маршрутизації сторінок та управління кошиком покупок, які забезпечують стабільну взаємодію через RESTful API. Розроблений інтерфейс користувача відповідає сучасним тенденціям дизайну ігрових платформ, поєднуючи естетичну привабливість з функціональністю через темне кольорове оформлення, інтуїтивну навігацію та адаптивне розташування елементів.

Проведено комплексне тестування всіх створених програмних модулів клієнтської архітектури з перевіркою системи при введенні як коректних, так і некоректних даних. Результати тестування підтвердили стабільність та надійність розробленого рішення. Створено детальну інструкцію користувача, що забезпечує дієву експлуатацію програмної системи.

Результати дослідження мають значну практичну цінність для індустрії розробки e-commerce платформ у сфері цифрових ігор. Розроблені архітектурні рішення можуть бути адаптовані та використані для створення аналогічних систем, а впроваджені принципи модульності та масштабованості становлять основу для подальших досліджень у цій галузі.

Отримані результати підтверджують дієвість обраного підходу до розробки клієнтської архітектури e-commerce платформи та створюють надійну основу для реалізації повнофункціональної системи цифрової дистрибуції комп'ютерних ігор, здатної конкурувати з існуючими рішеннями на ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Business Research [Електронний ресурс] – Режим доступу до ресурсу: https://www.businessresearchinsights.com/market-reports/digital-games-market-117464 (дата звернення: 12.03.2025)

2. Рельке А.А. Бабюк Н.П. Застосування алгоритмів машинного навчання для оптимізації веб-додатків. Матеріали IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня 2024 р. - Одеса, Видавництво ОНТУ, 2024 р. – С. 324-325

3. Дмитрієв В.Г., Рельке А.А. Технології створення e-comerce платформ/ Інформаційні технології- 2025 / Матеріали XII Всеукраїнської науково-практичної конференції молодих учених, Київ, 15травня 2025року - Київ, Видавництво ОНТУ, 2025 р. – С. 122-124

4. Codez Up[Електронний ресурс] – Режим доступу до ресурсу: https://codezup.com/designing-a-highly-available-e-commerce-platform-with-docker-and-kubernetes/ (дата звернення: 18.03.2025)

5. Shashanknathe [Електронний ресурс] – Режим доступу до ресурсу: https://shashanknathe.com/blog/desktop-app-with-react-electron-shadcn (дата звернення: 18.03.2025)

6. Finances online [Електронний ресурс] – Режим доступу до ресурсу: https://financesonline.com/steam-statistics/ (дата звернення: 18.03.2025)

7. Prizee [Електронний ресурс] – Режим доступу до ресурсу: https://www.prizee.com/epic-games-store-une-retrospective-2024-marquee-par-une-croissance-record/ (дата звернення: 28.03.2025)

8. Newzoo [Електронний ресурс] – Режим доступу до ресурсу: https://newzoo.com/resources/trend-reports/newzoos-global-games-market-report-2024-free-version (дата звернення: 08.04.2025)

9. Mordorinte lligence [Електронний ресурс] – Режим доступу до ресурсу: https://www.mordorintelligence.com/industry-reports/pc-game-market (дата звернення: 08.04.2025)

10. Logrusit [Електронний ресурс] – Режим доступу до ресурсу: https://games.logrusit.com/en/news/game-industry-trends/ (дата звернення: 08.04.2025)

11. Reanin [Електронний ресурс] – Режим доступу до ресурсу: https://www.reanin.com/reports/global-pc-games-market (дата звернення: 08.04.2025)

12. Andela [Електронний ресурс] – Режим доступу до ресурсу: https://www.andela.com/blog-posts/structuring-your-react-application-atomic-design-principles (дата звернення: 08.04.2025)

13. Dev [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/arishn/how-i-built-a-full-stack-mern-e-commerce-website-react-redux-nodejs-express-mongodb-3k5g (дата звернення: 08.04.2025)

14. Nitorin fotech [Електронний ресурс] – Режим доступу до ресурсу: https://www.nitorinfotech.com/blog/building-scalable-user-interfaces-with-atomic-design-in-react/ (дата звернення: 08.04.2025)

15. Theretailexec [Електронний ресурс] – Режим доступу до ресурсу: https://theretailexec.com/analytics/ecommerce-analytics-guide/ (дата звернення: 08.04.2025)

16. Mayple [Електронний ресурс] – Режим доступу до ресурсу: https://www.mayple.com/resources/ecommerce/ecommerce-analytics-tools (дата звернення: 08.04.2025)

17. Shopify [Електронний ресурс] – Режим доступу до ресурсу: https://www.shopify.com/enterprise/blog/ecommerce-data-analysis (дата звернення: 08.04.2025)

18. Bigcommerce [Електронний ресурс] – Режим доступу до ресурсу: https://www.bigcommerce.com/articles/ecommerce/ecommerce-analytics/ (дата звернення: 08.04.2025)

19. Geteppo [Електронний ресурс] – Режим доступу до ресурсу: https://www.geteppo.com/blog/product-usage (дата звернення: 08.04.2025)

20. Amplitude [Електронний ресурс] – Режим доступу до ресурсу: https://amplitude.com/explore/analytics/ecommerce-analytics-complete-guide (дата звернення: 08.04.2025)

21. Thelilipath [Електронний ресурс] – Режим доступу до ресурсу: https://thelilipath.com/analysis/ecommerce-analytics-use-cases/ (дата звернення: 08.04.2025)

22. Scaler [Електронний ресурс] – Режим доступу до ресурсу: https://www.scaler.com/topics/react/electron-react/ (дата звернення: 08.04.2025)

23. Codiga [Електронний ресурс] – Режим доступу до ресурсу: https://www.codiga.io/blog/build-electron-typescript-react-app/ (дата звернення: 08.04.2025)

24. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с

ДОДАТКИ

60

Додаток А – Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., професор О. Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка e-commerce платформи для комп'ютерних ігор. Частина 1. Архітектура клієнта» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доцент Н. П. Бабюк
«24» березня 2025 р.
Виконав:
студент гр. 5ПІ-21б А. А. Рельке
«24» червня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка e-commerce платформи для комп’ютерних ігор. Частина 1. Архітектура клієнта».

Галузь застосування – індустрія комп’ютерних ігор: платформи електронної комерції для продажу та розповсюдження ігор.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є підвищення функціонування e-commerce платформ для комп'ютерних ігор за рахунок розробки оптимізованої архітектури клієнтської частини, що забезпечує швидкодію, зручність використання та масштабованість системи.

Призначення роботи – розробка e-commerce платформи для ігор.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Рельке А.А. Бабюк Н.П. Застосування алгоритмів машинного навчання для оптимізації веб-додатків. Матеріали IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня 2024 р. - Одеса, Видавництво ОНТУ, 2024 р. – С. 324-325

2. Дмитрієв В.Г., Рельке А.А. Технології створення e-comerce платформ/ Інформаційні технології- 2025 / Матеріали XII Всеукраїнської науково-практичної конференції молодих учених, Київ, 15травня 2025року - Київ, Видавництво ОНТУ, 2025 р. – С. 122-124

5. Технічні вимоги

Комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 1 ГБ, стабільне з’єднання з інтернетом.

6. Конструктивні вимоги.

Десктопний застосунок та веб-клієнт повинні відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану e-commerce платформи для комп'ютерних ігор	24.03.25-30.03.25
2	Аналіз аналогів e-commerce платформи для комп'ютерних ігор	02.04.25-16.04.25
3	Розробка алгоритмів та модулів e-commerce платформи	16.04.25-21.04.25
4	Тестування e-commerce платформи	22.04.25-17.05.24
5	Оформлення матеріалів до захисту БКР	24.03.25-30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

63

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка e-commerce платформи для комп'ютерних ігор. Частина 1. Архітектура клієнта

Тип роботи: ____ бакалаврська кваліфікаційна робот
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: ІПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism ___8,9___ %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____доцент Бабюк Н. П._____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____Рельке А. А._____
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

```
import React, { useEffect, useState } from 'react';
import Api from './services/commersApiService';
import { Swiper, SwiperSlide } from "swiper/react";
import { Navigation, Pagination } from "swiper/modules";
import "swiper/css";
import "swiper/css/navigation";
import "swiper/css/pagination";

function ProductDetails({ productId, onBack }) {
   const [product, setProduct] = useState(null);
   const [error, setError] = useState(null);

   useEffect(() => {
      const fetchProduct = async () => {
         try {
            const api = Api();
            const response = await api.Product.GetProduct(productId);
            if (response) {
               setProduct(response);
            } else {
               throw new Error('Товар не найден');
            }
         } catch (err) {
            setError('Ошибка при загрузке данных');
            console.error(err);
         }
      };

      if (productId) {
         fetchProduct();
      }
   }, [productId]);

   //
   const handleAddToCart = async () => {
      try {
         const api = Api();
         await api.post('/cart/add', { productId });

         alert('Товар додано до кошика!');
      } catch (error) {
         console.error('Помилка при доданні до кошика:', error);

         alert('Помилка при доданні до кошика.');
      }
   };

   if (error) {
      return <div>Помилка: {error}</div>;
```

```
}

if (!product) {
   return <div>Завантаження...</div>;
}

return (
   <div className="product-details">
      <h1 className="product-details-name">{product?.productName}</h1>
      <div className="product-details-main">
         <div className="product-details-imgContainer">
            {product?.productMedia?.length > 0 && (
               <Swiper
                  modules={[Navigation, Pagination]}
                  navigation
                  pagination={{ clickable: true }}
                  loop={true}
                  className="product-slider"
               >
                  {product.productMedia.map((media, index) => (
                     <SwiperSlide key={index}>
                        <img
                           src={media.src}
                           alt={media.alt || "Product image"}
                           className="product-image"
                        />
                     </SwiperSlide>
                  ))}
               </Swiper>
            )}
         </div>
         <div className="product-details-txtContainer">
            <p>{product?.shortDescription}</p>

            {product?.developer?.length > 0 && (
               <p>Розробник: {product.developer.map(dev => dev.name).join(", ")}</p>
            )}

            {product?.publisher?.length > 0 && (
               <p>Видавець: {product.publisher.map(pub => pub.name).join(", ")}</p>
            )}

            {product?.categories?.[0] && (
               <p>Категорія: {product.categories[0].tytle}</p>
            )}

            <p>Дата випуска: {product?.releaseDate || "Неизвестно"}</p>

         </div>
      </div>
      <div className="product-details-containerPurchase">
         <h3>{product?.productName}</h3>
```

```
            <div className="product-details-containerPrice">
              <p>{product.price[0].value} {product.price[0].currency}</p>
              <button className="product-details-buttonBuy" onClick={handleAddToCart}>
                До кошика
              </button>

            </div>
          </div>
          <h1 className='product-details-configuration'>СИСТЕМНІ ВИМОГИ</h1>
          <div className="product-details-containerConfiguration">
            <div className="product-details-minimalConfiguration">
              <p>МІНІМАЛЬНІ:</p>
              <p>ОС: {product.configuration[0].minimal[0].OS}</p>
              <p>Процесор: {product.configuration[0].minimal[0].CPU}</p>
              <p>Оперативна пам’ять: {product.configuration[0].minimal[0].RAM}</p>
              <p>Відеокарта: {product.configuration[0].minimal[0].GPU}</p>
              <p>Місце на диску: {product.configuration[0].minimal[0].place}</p>
            </div>
            <div className="product-details-recommendedConfiguration">
              <p>РЕКОМЕНДОВАНІ:</p>
              <p>ОС: {product.configuration[0].recommended[0].OS}</p>
              <p>Процесор: {product.configuration[0].recommended[0].CPU}</p>
              <p>Оперативна пам’ять: {product.configuration[0].recommended[0].RAM}</p>
              <p>Відеокарта: {product.configuration[0].recommended[0].GPU}</p>
              <p>Місце на диску: {product.configuration[0].recommended[0].place}</p>
            </div>
          </div>
        </div>

    );
}

export default ProductDetails;



import React, { useEffect, useState } from 'react';
import Api from './services/commersApiService';

function ProductsCart({ onSelectProduct }) {
    const [products, setProducts] = useState([]);
    const [person, setPerson] = useState(null);
    const [error, setError] = useState(null);

    useEffect(() => {
        try {
            const api = Api();

            const productResponse = api.Product.GetCartProducts();
            if (productResponse && Array.isArray(productResponse.productItems)) {
                setProducts(productResponse.productItems);
            } else {
```

```
            throw new Error('Неправильний формат данних товарів');
        }

        const personResponse = api.Product.GetPersonInfo();
        if (personResponse && Array.isArray(personResponse.personItems)) {
            setPerson(personResponse.personItems[0]);
        } else {
            throw new Error('Неправильний формат данних користувача');
        }
    } catch (err) {
        setError('Помилка при завантаженні данних');
        console.error(err);
    }
}, []);

if (error) {
    return <div>Помилка: {error}</div>;
}

return (
    <div style={{ display: 'flex', flexDirection: 'column', gap: '20px' }}>
        {person && (
            <div style={{
                display: 'flex',
                alignItems: 'center',
                background: '#333',
                color: '#fff',
                padding: '15px',
                borderRadius: '10px',
                textAlign: 'center',
                maxWidth: '300px'
            }}>
                <img
                    src={person.personImg[0].src}
                    alt={person.personImg[0].alt}
                    style={{ width: '60px', height: '60px', borderRadius: '50%', marginRight: '15px' }}
                />
                <div>
                    <h3 style={{ fontSize: '18px', margin: '0' }}>{person.personName}</h3>
                    <p style={{ fontSize: '14px', color: '#bbb' }}>{person.shortInfo}</p>
                </div>
            </div>
        )}

        {/* Блок товаров */}
        {products.map((product) => (
            <div key={product.Id} className='cart-container-product'>
                <img
                    src={product.previewProductMedia[0].src}
                    alt={product.previewProductMedia[0].alt}
                    style={{ width: '100%', borderRadius: '10px' }}
                />
```

```
                <h3 style={{ fontSize: '16px', margin: '10px 0' }}>{product.productName}</h3>
                <button className='cart-container-button'>
                  Завантажити
                </button>
              </div>
            ))}
        </div>
    );
};

export default ProductsCart;


import React, { useEffect, useState } from 'react';
import Api from './services/commersApiService';

function ProductsList({ onSelectProduct }) {
    const [products, setProducts] = useState([]);
    const [error, setError] = useState(null);

    useEffect(() => {
        const fetchProducts = async () => {
            try {
                const api = Api();
                const response = api.Product.GetProducts();
                if (response && Array.isArray(response.productItems)) {
                    setProducts(response.productItems);
                } else {
                    throw new Error('Неправильный формат данных');
                }
            } catch (err) {
                setError('Ошибка при загрузке данных');
                console.error(err);
            }
        };

        fetchProducts();
    }, []);

    if (error) {
        return <div>Ошибка: {error}</div>;
    }

    return (
        <div>
            <h1 className='main-title'>Список продуктів</h1>
            <div className="products-list">
                {products.map((product) => (
                    <div key={product.Id} className="product-item">
                        <div className="product-item-visible" onClick={() =>
onSelectProduct(product.Id)}>
                            <img
```

```
                        src={product.previewProductMedia[0]?.src}
                        alt={product.previewProductMedia[0]?.alt || 'Product image'}
                        style={{ width: '400px', height: 'auto' }}
                    />
                    <div className="product-list-mainInformation">
                        <p className='product-list-gameName'>{product.productName}</p>
                        <p className='product-list-price'>
                            {product.price[0].value} {product.price[0].currency}
                        </p>
                    </div>
                </div>
                <div className="product-item-hidden">
                    <img
                        src={product.previewProductMedia[0]?.src}
                        alt={product.previewProductMedia[0]?.alt || 'Product image'}
                        style={{ width: '200px', height: 'auto' }}
                    />
                    <p>{product.shortDescription}</p>
                    <p>Дата випуску: {product.releaseDate}</p>
                </div>
            </div>
        ))}
      </div>
    </div>
  );
}

export default ProductsList;


import React, { useState, useEffect } from 'react';
import './App.css';
import ProductsList from './productsList';
import ProductDetails from './productDetails';
import ProductsCart from './productsCart';

function App() {
  const [selectedProductId, setSelectedProductId] = useState(null);
  const [content, setContent] = useState('shop');

  useEffect(() => {
    const shopButton = document.getElementById('shop-button');
    const cartButton = document.getElementById('personCart');

    const handleShopClick = () => {
      setContent('shop');
      setSelectedProductId(null);
    };

    const handleCartClick = () => {
      setContent('cart');
      setSelectedProductId(null);
```

```
        };

        if (shopButton) shopButton.addEventListener('click', handleShopClick);
        if (cartButton) cartButton.addEventListener('click', handleCartClick);

        return () => {
            if (shopButton) shopButton.removeEventListener('click', handleShopClick);
            if (cartButton) cartButton.removeEventListener('click', handleCartClick);
        };
    }, []);

    return (
        <div className="App">
            {content === 'cart' ? (
                <ProductsCart />
            ) : selectedProductId ? (
                <ProductDetails productId={selectedProductId} onBack={() =>
setSelectedProductId(null)} />
            ) : (
                <ProductsList onSelectProduct={setSelectedProductId} />
            )}
        </div>
    );
}

export default App;


/**
 * API Service constructor.
 * @param {string} baseUrl - Base URL for the API.
 * @returns {object} - Object containing HTTP methods.
 */
function apiService(baseUrl) {
    /**
     * Perform a GET request.
     * @param {string} path - API path.
     * @param {object} [params={}] - Query parameters as key-value pairs.
     * @returns {Promise<void>}
     */
    async function get(path, params = {}) {
        try {
            const url = new URL(baseUrl + path);
            Object.keys(params).forEach(key => url.searchParams.append(key, params[key]));

            const response = await fetch(url);
            if (!response.ok) {
                throw new Error('Network response was not ok ' + response.statusText);
            }
            const data = await response.json();
            document.getElementById('output').textContent = data.message;
        } catch (error) {
```

```
        console.error('There has been a problem with your fetch operation:', error);
    }
}

/**
 * Perform a POST request.
 * @param {string} path - API path.
 * @param {object} newData - Data to send in the request body.
 * @returns {Promise<void>}
 */
async function post(path, newData) {
    try {
        const response = await fetch(baseUrl + path, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(newData)
        });
        if (!response.ok) {
            throw new Error('Network response was not ok ' + response.statusText);
        }
        const data = await response.json();
        document.getElementById('output').textContent = data.message;
    } catch (error) {
        console.error('There has been a problem with your fetch operation:', error);
    }
}

/**
 * Perform a PUT request.
 * @param {string} path - API path.
 * @param {object} updatedData - Data to update in the request body.
 * @returns {Promise<void>}
 */
async function put(path, updatedData) {
    try {
        const response = await fetch(baseUrl + path, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(updatedData)
        });
        if (!response.ok) {
            throw new Error('Network response was not ok ' + response.statusText);
        }
        const data = await response.json();
        document.getElementById('output').textContent = data.message;
    } catch (error) {
        console.error('There has been a problem with your fetch operation:', error);
    }
```

```
    }

    return { get, post, put };
}

/**
 * API object with predefined controllers and methods.
 * @returns {object}
 */
function Api() {
    const baseUrl = window.env.CMS_DOMAIN;
    const api = apiService(baseUrl);

    /**
     * Product controller.
     */
    const Product = {
        ControllerUrl: `/products`,

        /**
         * Get a specific product.
         * @param productId - Product ID (only 1, 2, or 3 are allowed).
         * @returns {object} - Product details.
         */
        GetProduct: function (productId) {
            const action = "/products";
            return {
                Id: 1,
                productName: "Black Myth: Wukong",
                productMedia: [
                    {
                        type: "image", //allowed type: image, video
                        src:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                        alt: "wukong",
                    }
                ],
                shortDescription: "Black Myth: Wukong is an action RPG rooted in Chinese mythology.
You shall set out as the Destined One to venture into the challenges and marvels ahead, to uncover
the obscured truth beneath the veil of a glorious legend from the past.",
                releaseDate: "20 серп. 2024 г.",
                developer: [{
                    id: 1,
                    name: "Game Science",
                }],
                publisher: [{
                    id: 1,
                    name: "Game Science",
                }],
                categories: [
                    {
```

```
                id: 1,
                tytle: "MMythology",
            }
        ],
        price: [
            {
                culture: "eu",
                value: 2.1,
                currency: "EUR",
            }
        ],
        configuration: [
            {
                minimal: [
                    {
                        OS: "Windows 10 64-bit",
                        CPU: "Intel Core i5-8400 / AMD Ryzen 5 1600",
                        RAM: "16 GB ОЗУ",
                        GPU: "NVIDIA GeForce GTX 1060 6GB / AMD Radeon RX 580 8GB",
                        place: "130 GB",
                    }
                ],
                recommended: [
                    {
                        OS: "Windows 10 64-bit",
                        CPU: "Intel Core i7-9700 / AMD Ryzen 5 5500",
                        RAM: "16 GB ОЗУ",
                        GPU: "NVIDIA GeForce RTX 2060 / AMD Radeon RX 5700 XT / INTEL Arc
A750",
                        place: "130 GB",
                    }
                ]
            }
        ]
    };
},

/**
 * Get all products.
 * @returns {void}
 */
GetProducts: function () {
    const action = "/products";
    return {
        productItems: [{
            Id: 1,
            productName: "Black Myth: Wukong",
            previewProductMedia: [
                {
                    type: "image", //allowed type: image, video
```

```
                    src:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                    alt: "wukong",
                }
            ],
            shortDescription: "Black Myth: Wukong is an action RPG rooted in Chinese
mythology.",
            releaseDate: "20 серп. 2024 г.",
            categories: [
                {
                    id: 1,
                    tytle: "",
                }
            ],
            price: [
                {
                    culture: "eu",
                    value: 2.1,
                    currency: "EUR",
                }
            ]
        }]

    }

    //return api.get(this.ControllerUrl + action, {});
},

/**
 * Get filtered products.
 * @returns {void}
 */
GetFilteredProducts: function () {
    const action = "/filteredProducts";
    return {
        productItems: [{
            Id: 1,
            productName: "Tower",
            previewProductMedia: [
                {
                    type: "image", //allowed type: image, video
                    src:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                    alt: "wukong",
                }
            ],
            shortDescription: "Black Myth: Wukong is an action RPG rooted in Chinese
mythology. ",
            releaseDate: "20 серп. 2024 г.",
            categories: [
```

```
                {
                    id: 1,
                    tytle: "",
                }
            ],
            price: [
                {
                    culture: "eu",
                    value: 2.1,
                    currency: "EUR",
                }
            ]
        }]
    }
    //return api.get(this.ControllerUrl + action, {});
},

/**
 * Get cart products.
 * @returns {void}
 */

GetCartProducts: function () {
    const action = "/cartProducts";
    return {
        productItems: [{
            Id: 1,
            productName: "Tower",
            previewProductMedia: [
                {
                    type: "image", //allowed type: image, video
                    src:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                    alt: "wukong",
                }
            ],
        }
        ]
    }
},
/**
* Get person info.
* @returns {void}
*/

GetPersonInfo: function () {
    const action = "/person";
    return {
        personItems: [{
            Id: 1,
            personName: "Vorsinok",
```

```
                personImg: [
                    {
                        type: "image", //allowed type: image, video
                        src:
"https://i.pinimg.com/736x/0e/d0/b1/0ed0b1aadf7a7d1ce8ced1aace20daea.jpg",
                        alt: "avatar",
                    }
                ],
                shortInfo: "hi!",
            }]
        }
    }

};

//post замокать обьект
const Cart = {
    AddToCart: function () {
        //return api.post('/cart/add', { productId });
        const action = "/addToCart";
        return {
            productItems: [{
                Id: 1,
                productName: "Tower",
                previewProductMedia: [
                    {
                        type: "image", //allowed type: image, video
                        src:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                        alt: "wukong",
                    }
                ],
            }
            ]
        }
    }
};

/**
 * Category controller.
 */
const Category = {
    ControllerUrl: `/category`,

    /**
     * Get a category.
     * @returns {void}
     */
    GetCategory: function () {
        const action = "/category";
        //return api.get(this.ControllerUrl + action, {});
```

```
            return {
                name: "ЛИДЕРЫ ПРОДАЖ",
                description: "Все продукты",
            }
        },
    };

    /**
     * Search controller.
     */
    const Search = {
        ControllerUrl: `/search`,

        /**
         * Perform a search and return results.
         * @returns {object} - Search results.
         */
        GetCategory: function () {
            const action = "/search";
            return {
                items: [
                    {
                        name: "somecategoryname",
                        img:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                        id: 1,
                        type: "product or category",
                    },
                    {
                        name: "somesecondcategoryname",
                        img:
"https://shared.fastly.steamstatic.com/store_item_assets/steam/apps/2358720/header.jpg?t=1734426
465",
                        id: 1,
                        type: "product or category",
                    }
                ]
            };
        },
    };

    return { Product, Category, Search, Cart };
}

export default Api;
```

Додаток Г – Ілюстративна частина

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота на тему:
«Розробка e-commerce платформи для комп'ютерних ігор.
Частина 1. Архітектура Клієнта»

Автор: ст. гр. 5ПІ-21б
Рельке А. А.
Науковий керівник: К.Т.Н.,
доцент ПЗ Бабюк Н. П.

Вінниця - 2025

Рисунок Г.1 – Титульний слайд

Мета, предмет та об’єкт дослідження

- Метою дослідження є підвищення дієвості функціонування e-commerce платформ для комп'ютерних ігор за рахунок розробки оптимізованої архітектури клієнтської частини, що забезпечує швидкодію, зручність використання та масштабованість системи.
- Об’єктом дослідження є процес функціонування клієнтської частини e-commerce платформ для комп'ютерних ігор з використанням сучасних веб-технологій.
- Предметом дослідження є методи та засоби розробки архітектури клієнтських модулів e-commerce платформи для комп'ютерних ігор з використанням React та супутніх технологій.

Рисунок Г.2 – Мета, об’єкт та предмет дослідження

Рисунок Г.3 – Актуальність розробки

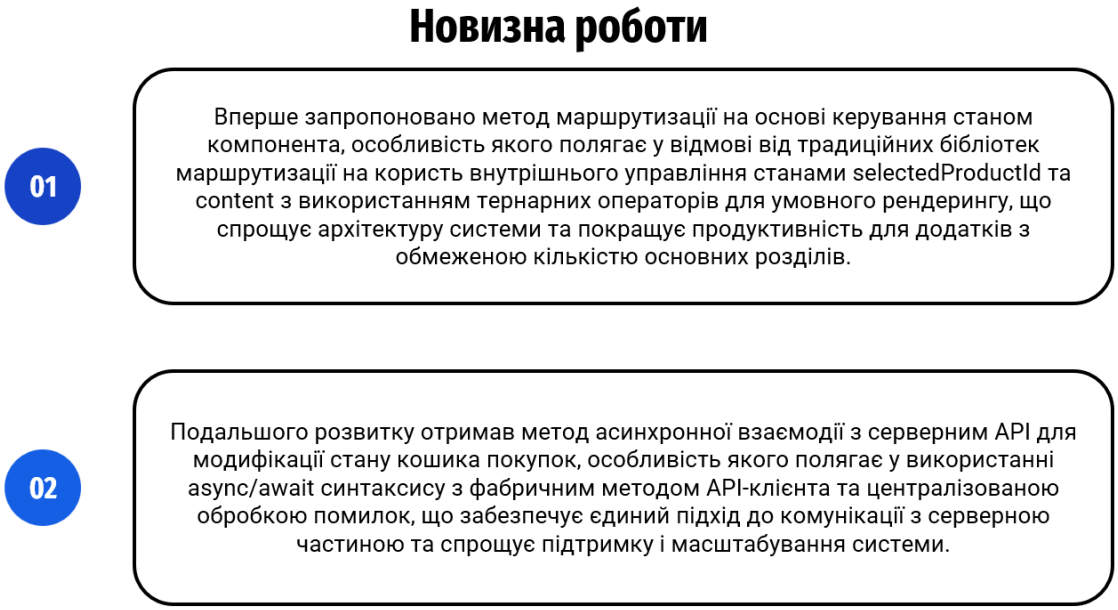

Рисунок Г.4 – Новизна роботи. Частина 1

Новизна роботи

03

Вперше запропоновано метод динамічного управління навігаційними переходами в e-commerce системах, особливість якого полягає у комбінуванні реактивного програмування з адаптивною валідацією контенту та каскадним оновленням залежних компонентів, що забезпечує плавні переходи між розділами платформи та покращує користувацький досвід взаємодії з інтерфейсом.

04

Подальшого розвитку отримав метод асинхронного отримання даних з серверної частини, особливість якого полягає у використанні React Hooks з інтегрованою валідацією структури даних та обробкою помилок через try-catch конструкції, що дозволяє забезпечити надійність додатку та реактивне оновлення інтерфейсу при зміні даних.

Рисунок Г.5 – Новизна роботи. Частина 2

Порівняльний аналіз аналогів

	Steam	Epic Game	GOG	БДР
Комісія для розробників	30%	20%	12%	10%
Крос-платформність	+	+/-	-	+
Мобільна підтримка	+/-	+/-	-	+/-
Інтеграція з блокчейн	-	+/-	-	+/-
Інтеграція з соцмережами	-	+	+	+
Підтримка інді-розробників	+/-	+	-	+
DRM	+	+/-	-	+

Рисунок Г.6 – Порівняльний аналіз аналогів

Вибір засобів для реалізації застосунку

Технологія/Мова	Продуктивність	Швидкість розробки	Екосистема	Гнучкість
JavaScript (React, Vue, Angular)	★★★★★	★★★★★	★★★★★	★★★★★
Python (через Django Templates)	★★	★★★	★★	★★
PHP (через Blade, Twig тощо)	★★★	★★★★	★★★	★★★
Java (Thymeleaf, JSP)	★★★★	★★	★★★	★★★
Ruby (ERB, HAML)	★★★	★★★	★★	★★★

Рисунок Г.7 – Вибір засобів для реалізації застосунку

Загальний алгоритм роботи системи

Рисунок Г.8 – Загальний алгоритм системи

Рисунок Г.9 – Алгоритм отримання продуктів з сервера

Рисунок Г.10 – Алгоритм маршрутизації сторінок

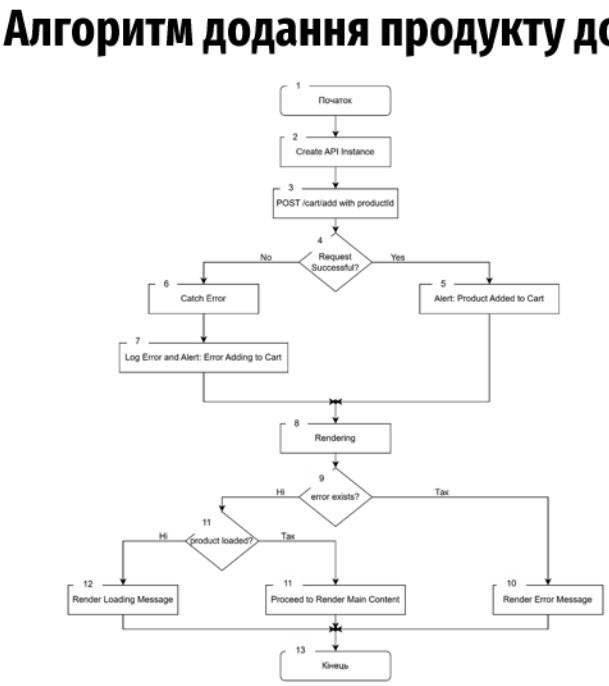

Рисунок Г.11 – Алгоритм додання продукту до кошика

Рисунок Г.12 – Використані технології

Демонстрації інтерфейсу застосунку «Реєстрація та авторизація»

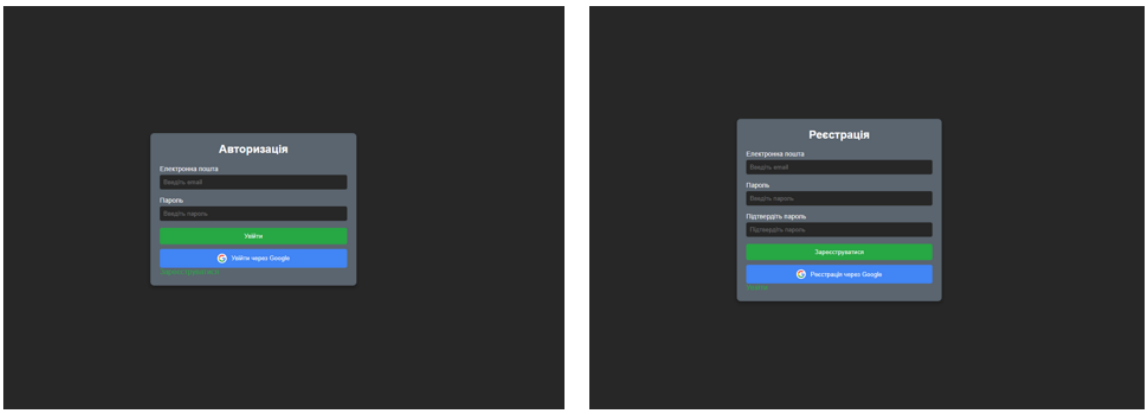

Рисунок Г.13 – Демонстрації інтерфейсу застосунку «Реєстрація та авторизація»

Демонстрації інтерфейсу застосунку «Головний екран»

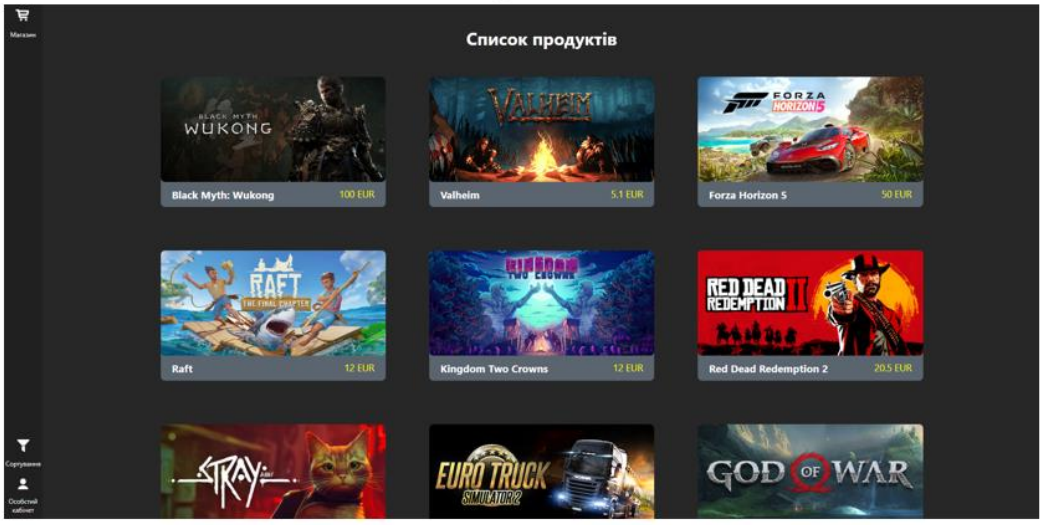

Рисунок Г.14 – Демонстрації інтерфейсу застосунку «Головний екран»

Демонстрації інтерфейсу застосунку «Сторінка гри»

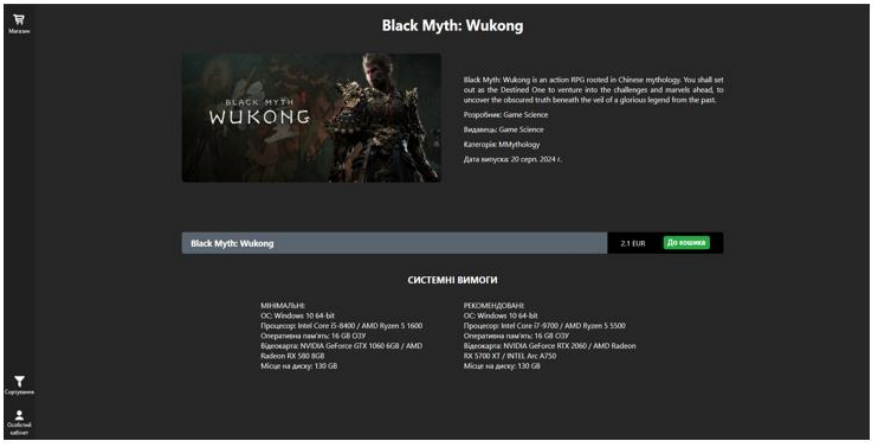

Рисунок Г.15 – Демонстрації інтерфейсу застосунку «Сторінка гри»

Демонстрації інтерфейсу застосунку «Особистий кабінет користувача»

Рисунок Г.16 – Демонстрації інтерфейсу застосунку «Особистий кабінет користувача»

Результат розробки

- ✓ **Проаналізовано стан сучасних підходів до розробки e-commerce платформ** для цифрових продуктів з використанням React технологій та обґрунтовано необхідність розробки спеціалізованих модулів для комп'ютерних ігор;
- ✓ **розроблено архітектуру клієнтської частини e-commerce системи** з модульною структурою компонентів для каталогу продуктів, деталей товарів та кошика покупок;
- ✓ **розроблено алгоритми динамічного управління навігацією** та обробки користувацьких переходів, а також алгоритми асинхронного отримання даних з серверної частини;
- ✓ **обґрунтовано вибір засобів для реалізації** e-commerce платформи з використанням React Hooks, сучасних JavaScript практик та RESTful API;
- ✓ **розроблено інтерактивні компоненти користувацького інтерфейсу** з використанням React фреймворку для відображення каталогу ігор, деталей продуктів та функціоналу кошика;
- ✓ **проведено тестування програмних модулів** навігації, отримання даних та взаємодії з API, підтверджено ефективність реалізованих методів маршрутизації та управління станом.

Рисунок Г.17 – Результати розробки

Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

1. Рельке А.А. Бабюк Н.П. Застосування алгоритмів машинного навчання для оптимізації веб-додатків. Матеріали IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня 2024 р. - Одеса, Видавництво ОНТУ, 2024 р. – С. 324-325
2. Дмитрієв В.Г., Рельке А.А. Технології створення e-comerce платформ/ Інформаційні технології- 2025 / Матеріали XII Всеукраїнської науково-практичної конференції молодих учених, Київ, 15травня 2025року - Київ, Видавництво ОНТУ, 2025 р. – С. 122-124

Рисунок Г.18 –Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Дякую за увагу!

Рисунок Г.19 – Фінальний слайд