

***БАКАЛАВРСКА РОБОТА***

**БР.ІІ – 63.00.00.000 ІІЗ**

**Група ІІ-22-3**

**Гевкач Денис**

**Івано-Франківський національний технічний університет нафти і газу**

**Факультет інформаційних технологій**

**Кафедра інженерії програмного забезпечення**

**Гевкач Денис Борисович**

---

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

## **БАКАЛАВРСЬКА РОБОТА**

**Розроблення інтелектуального веб-сервісу для монетизації та  
персоналізації ігрового досвіду**

(назва роботи)

**Інженерія програмного забезпечення**

---

(назва освітньої програми)

**121– Інженерія програмного забезпечення**

---

(шифр і назва спеціальності)

**Робота містить результати власних досліджень, використання ідей, результатів і  
текстів інших авторів мають посилання на відповідне джерело:**

Здобувач освітнього ступеня Гевкач Д. Б.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Зікратий Серій Вікторович, к.т.н. доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

**Допущено до захисту**

**Завідувач кафедри**

доц. Бандура В.В.

(посада)  
прізвище)

(підпис) (дата) (ініціали та

**Івано-Франківськ – 2026**

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ:**

Зав. кафедрою ІПЗ

доц. В.В. Бандура

“ ” 2026 р.

## **ЗАВДАННЯ**

### **НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ**

**Гевкачу Денису Борисовичу**

(прізвище, ім'я, по-батькові)

**1. Тема проекту (роботи) “Розроблення інтелектуального веб-сервісу для монетизації та персоналізації ігрового досвіду”**

керівник проекту (роботи) Зікратий Серій Вікторович, д.т.н., доцент

затверджені наказом закладу вищої освіти від “ 21 ” квітня 2026 р. № 179/7

**2. Строк подання студентом проекту (роботи) 2026 р.**

**3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики**

**4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)**

1. Аналіз теорії конструювання програмного забезпечення

2. Просування бази даних системи

3. Опис бізнес-логіки

4. Представлення інтерфейсу програмного продукту

5. Тестування програмного продукту

**5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)**

1. Інформаційні потоки між компонентами системи (рис. 1.1, ст. 24)

2. Матриця пріоритетів вимог (рис. 2.1, ст. 31)

3. Детальна архітектура сервісів і інтеграцій (рис. 3.1, ст. 42)

4. Деталізована діаграма компонентів з підсистемами (рис. 3.2, ст. 49)

5. Панель покупки шкіни (підсвічено вже куплений шкіни) (рис. 4.3, ст. 57)

**1. Консультанти розділів проекту (роботи)**

| <i>Розділ</i> | <i>Консультант</i> | <i>Підпис, дата</i>   |                         |
|---------------|--------------------|-----------------------|-------------------------|
|               |                    | <i>Завдання видав</i> | <i>Завдання прийняв</i> |
|               |                    |                       |                         |
|               |                    |                       |                         |
|               |                    |                       |                         |

**2. Дата видачі завдання \_\_\_\_\_**

*Керівник*

\_\_\_\_\_  
(підпис)

Зікратий С. В.

(розшифровка підпису)

*Завдання прийняв до виконання*

\_\_\_\_\_  
(підпис)

Гевкач Д. Б.

(розшифровка підпису)

***КАЛЕНДАРНИЙ ПЛАН***

| <i>№ з/п</i> | <i>Назва етапів дипломного проекту (роботи)</i>        | <i>Строк виконання етапів проекту</i> | <i>Примітка</i> |
|--------------|--------------------------------------------------------|---------------------------------------|-----------------|
| 1            | Опрацювання теоретичних матеріалів з теми роботи       | 27.02.2026                            | виконано        |
| 2            | Вибір інструментів, технологій та методів розробки     | 05.03.2026                            | виконано        |
| 3            | Налаштування середовища та конфігурація бази даних     | 23.03.2026                            | виконано        |
| 4            | Розробка компонентів та функціональних модулів системи | 13.04.2026                            | виконано        |
| 5            | Тестування системи та усунення виявлених помилок       | 23.04.2026                            | виконано        |
| 6            | Підготовка та оформлення бакалаврської роботи          | 27.05.2026                            | виконано        |

*Студент – дипломник*

\_\_\_\_\_  
(підпис)

Гевкач Д. Б.

(розшифровка підпису)

*Керівник роботи*

\_\_\_\_\_  
(підпис)

Зікратий С. В.

(розшифровка підпису)

## АНОТАЦІЯ

Бакалаврська робота містить 73 сторінки, 15 рисунків, список використаних джерел із 50 найменуваннями.

**Метою роботи** розробка інтелектуального веб-сервісу для монетизації та персоналізації ігрового досвіду, що забезпечує надійну обробку транзакцій, механізми персоналізації.

**Об'єкт дослідження:** процеси розробки та функціонування веб-сервісів для монетизації цифрового контенту в ігрових проєктах.

**Предмет дослідження:** методи, технології та інструменти реалізації модулів монетизації та інтеграції з платіжним провайдером і клієнтом.

**Результати дослідження:** реалізовано прототип серверної частини на Java з набором REST-ендпоінтів для обробки транзакцій

**В першому розділі** проаналізовано сучасні підходи до монетизації і персоналізації, оглянуто релевантні технології та нормативні аспекти.

**В другому розділі** сформульовано вимоги, описано модель даних і постановку задачі проєктування.

**В третьому розділі** наведено архітектуру системи, описано компоненти, API-контракти та інтеграційні сценарії з Unity і платіжними провайдерами.

**В четвертому розділі** описано реалізацію модулів: платіжний модуль зі Stripe, магазин скінів, адміністративну панель, модуль відгуків з інтеграцією Hugging Face і візуалізацію даних через Chart.js.

**У п'ятому розділі** представлено результати тестування, валідацію механізмів idempotency і конкурентного списання.

**Висновок:** реалізований прототип підтвердив можливість швидкого запуску базової платформи монетизації з прозорою обробкою платежів

**КЛЮЧОВІ СЛОВА:** ВЕБ-СЕРВІС, МОНЕТИЗАЦІЯ, ПЕРСОНАЛІЗАЦІЯ, РЕКОМЕНДАЦІЙНИЙ МОДУЛЬ, ТРАНЗАКЦІЇ, REST-API, JWT, STRIPE, DOCKER COMPOSE.

## ANNOTATION

This bachelor's thesis consists of 73 pages, 15 figures, a bibliography of 50 references.

**The aim of the thesis** is to develop an intelligent web service for monetizing and personalizing the gaming experience.

**Research object:** the processes of developing and operating web services for monetizing digital content in gaming projects.

**Research subject:** methods, technologies, and tools for implementing monetization modules, event tracking, recommendations, and integration with payment providers and the Unity client.

**Research results:** a prototype of the server-side component was implemented in Java with a set of REST endpoints for transaction processing.

**The first chapter** analyzes modern approaches to monetization and personalization, reviews relevant technologies, and discusses regulatory aspects.

**The second chapter** formulates functional and non-functional requirements, describes the data model, and outlines the design problem.

**The third chapter** presents the system architecture, describes the components, API contracts.

**The fourth chapter** describes the implementation of the modules: the payment module with Stripe, the skin store, the admin panel.

**The fifth chapter** presents the results of testing (functional, load, and integration), as well as the validation of idempotency and concurrent write mechanisms.

**Conclusion:** The implemented prototype confirmed the possibility of quickly launching a basic monetization platform.

**KEYWORDS:** WEB SERVICE, MONETIZATION, PERSONALIZATION, RECOMMENDATION MODULE, TRANSACTIONS, REST API, JWT, STRIPE, DOCKER COMPOSE.

## ЗМІСТ

|                                                                                                                             |           |
|-----------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ .....</b>                                                                                      | <b>8</b>  |
| <b>ВСТУП .....</b>                                                                                                          | <b>9</b>  |
| <b>РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНСТРУМЕНТІВ АНАЛІЗУ БЕЗПЕКИ ПРОГРАМНОГО КОДУ.....</b> | <b>11</b> |
| 1.1 Аналіз сучасного стану проблематики.....                                                                                | 11        |
| 1.2 Основні поняття та визначення інженерії ПЗ в даному домені.....                                                         | 13        |
| 1.3. Сучасні технології та тенденції, що застосовуються в області сервісних вебсайтів .....                                 | 16        |
| 1.4 Висновки по розділу.....                                                                                                | 23        |
| <b>РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ.....</b>                                                                     | <b>27</b> |
| 2.1 Аналіз вимог до веб-сервісу монетизації та персоналізації .....                                                         | 27        |
| 2.2 Формулювання функціональних та нефункціональних вимог.....                                                              | 29        |
| 2.3 Постановка задачі проектування системи.....                                                                             | 34        |
| 2.4 Висновки до розділу.....                                                                                                | 36        |
| <b>РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ.....</b>                                                                                  | <b>38</b> |
| 3.1 Розробка архітектури програмного забезпечення.....                                                                      | 38        |
| 3.2 Моделювання бізнес-процесів та системних компонентів.....                                                               | 39        |
| 3.3 Вибір та обґрунтування технологій та інструментів розробки.....                                                         | 45        |
| 3.4 Висновки по розділу.....                                                                                                | 47        |
| <b>РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ.....</b>                                                                                    | <b>48</b> |
| 4.1 Опис розробки програмного коду.....                                                                                     | 48        |
| 4.2 Використані алгоритми та структури даних.....                                                                           | 53        |
| 4.3 Модульне тестування та налагодження.....                                                                                | 56        |
| 4.4 Висновки по розділу.....                                                                                                | 58        |

|           |      |               |        |      |                                                                                                                         |                  |      |         |
|-----------|------|---------------|--------|------|-------------------------------------------------------------------------------------------------------------------------|------------------|------|---------|
|           |      |               |        |      | БР.ІП – 63.00.00.000 ПЗ                                                                                                 |                  |      |         |
| Змн.      | Арк. | № докум.      | Підпис | Дата | Розроблення інтелектуального веб-сервісу для монетизації та персоналізації ігрового досвіду<br><br>Пояснювальна записка | Лім.             | Арк. | Аркушів |
| Розроб.   |      | Гевкач Д.Б.   |        |      |                                                                                                                         |                  |      |         |
| Перевір.  |      | Зікратий С.В. |        |      |                                                                                                                         |                  |      |         |
| Реценз.   |      | Ваврик Т.О.   |        |      |                                                                                                                         |                  |      |         |
| Н. Контр. |      | Піх М.М.      |        |      |                                                                                                                         |                  |      |         |
| Затверд.  |      | Бандура В.В.  |        |      |                                                                                                                         | ІФНТУНГ, ІП-22-3 |      |         |

|                                                              |           |
|--------------------------------------------------------------|-----------|
| <b>РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ.....</b>             | <b>60</b> |
| 5.1 Стратегії та методи тестування.....                      | 60        |
| 5.2 Результати тестування системи та роботи її сервісів..... | 61        |
| 5.3 Аналіз ефективності впровадження.....                    | 63        |
| 5.4 Висновки по розділу.....                                 | 64        |
| <b>ВИСНОВКИ.....</b>                                         | <b>65</b> |
| <b>СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ.....</b>                         | <b>67</b> |
| <b>БІБЛІОГРАФІЧНА ДОВІДКА</b>                                |           |

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - інтерфейс прикладного програмування

REST - архітектурний стиль RESTful API

JSON - формат обміну даними JavaScript Object Notation

JWT - JSON Web Token для автентифікації та авторизації

СУБД - система управління базами даних

DB - база даних

ER - діаграма сутність-зв'язок (Entity-Relationship)

UML - мова моделювання Unified Modelling Language

ООП - об'єктно-орієнтоване програмування

SQL - мова структурованих запитів Structured Query Language

IAP - внутрішні покупки (In-App Purchases)

KPI - ключовий показник ефективності (Key Performance Indicator)

ML - машинне навчання

AI - штучний інтелект

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

## ВСТУП

Ігрова індустрія стрімко змінюється: монетизація перестала бути покупкою за гроші, а персоналізація - набором шаблонних налаштувань. Сьогодні гравець очікує від сервісу не лише коректної роботи платежів, а й певного підходу та пропозицій, швидкої реакції системи на поведінку.

**Актуальність теми** зумовлена потребою в інтелектуальних сервісах, які поєднують бізнес-логіку монетизації з механізмами персоналізації, забезпечують аналітику та дають змогу масштабуватися під навантаження.

У рамках бакалаврської роботи поставлено **завдання** створити прототип веб-сервісу, що реалізує базові функції обробки транзакцій, обліку балансу, збору відгуків та формування персоналізованих профілів.

Такий підхід дозволяє підвищити кількість пропозицій і покращити утримання користувачів.

**Об'єктом бакалаврської роботи** виступає інформаційна система для монетизації та персоналізації цифрового контенту.

**Предметом** є архітектура та реалізація веб-сервісу, що забезпечує обробку транзакцій, збір відгуків користувачів, збереження профілів та формування відповіді на основі простих алгоритмів.

**Мета бакалаврської роботи** - реалізувати інтелектуальний веб-сервіс, який поєднує механізми монетизації та персоналізації, та провести його первинну перевірку в умовах функціонального та навантажувального тестування

. Для досягнення мети поставлено такі завдання:

- проаналізувати сучасні підходи до монетизації цифрового контенту та методи персоналізації, опрацювати нормативні й технічні джерела;
- сформулювати функціональні та нефункціональні вимоги до веб-сервісу;
- спроектувати архітектуру системи, модель даних і API-контракти;
- реалізувати серверну частину прототипу на Java з набором

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

REST-ендпоінтів для обробки транзакцій і збору подій;

- впровадити простий рекомендаційний модуль на основі правил і статистичних метрик;
- провести модульні та навантажувальні тести, оцінити ключові показники ефективності.

**Методи**, що застосовувалися при виконанні роботи, включають аналіз літературних і технічних джерел, порівняльний аналіз існуючих рішень, проєктування архітектури з використанням UML-діаграм, моделювання інформаційних потоків, імплементацію прототипу з подальшим тестуванням.

**Новизна** отриманих результатів полягає в інтеграції базових механізмів монетизації з простим модулем відгуків у єдиному веб-сервісі, адаптованому для подальшої інтеграції з ігровим клієнтом. Запропоновано набір API-контрактів і модель даних, що забезпечують гнучкість при розширенні функціоналу.

**Практичне значення** отриманих результатів полягає в тому, що реалізований прототип може бути використаний як основа для подальшої інтеграції з ігровим сервісом або додатком.

**У висновку** виконаної роботи отримано робочий прототип веб-сервісу з базовим набором функцій монетизації та персоналізації, описано архітектуру та методи оцінки ефективності, виявлено обмеження реалізації і окреслено напрями подальшого розвитку.

Такий проєкт має перспективи на розширення та розвиток, отримання монетизації та значної підтримки аудиторії у випадку належних рекламних інтеграцій та колаборацій з іншими проєктами.

До вже зазначеного, доцільно включити в початкову фазу проєкту чіткий набір KPI для оцінки успішності прототипу (наприклад, конверсія покупки, середній чек, retention 7/30 днів, середній час обробки webhook-ів.

Бакалаврська робота містить 73 сторінки, 15 рисунків, список використаних джерел із 50 найменуваннями.

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

# РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНСТРУМЕНТІВ АНАЛІЗУ БЕЗПЕКИ ПРОГРАМНОГО КОДУ

## 1.1. Аналіз сучасного стану проблематики

Проблема поєднання монетизації та персоналізації ігрового досвіду в контексті веб-сервісів розглядається в літературі досить широко, але розподіл акцентів у вітчизняних і зарубіжних джерелах помітно відрізняється. У зарубіжних публікаціях основна увага приділяється практичним моделям монетизації, аналітиці поведінки користувачів і застосуванню методів машинного навчання для персоналізації контенту [48]. У вітчизняних роботах частіше зустрічаються огляди архітектурних рішень, питання безпеки та нормативно-правові аспекти обробки персональних даних, а також прикладні матеріали з розробки серверних компонентів [22]. Такий розподіл відображає реальні потреби індустрії: міжнародні компанії інвестують у масштабні аналітичні платформи й ML-моделі, тоді як локальні розробники зосереджуються на надійності, інтеграції та відповідності законодавству [30].

У зарубіжній літературі виділяються кілька напрямів, що мають пряме відношення до теми бакалаврської роботи. Перший напрям - моделі монетизації: аналізуються підходи IAP (in-app purchases), підпискові моделі, рекламні інтеграції та гібридні стратегії [48]. Описуються бізнес-метрики (ARPU, ARPPU, LTV, retention), методи їх вимірювання та оптимізації через A/B-тестування і персоналізовані пропозиції [13]. Другий напрям - персоналізація: розглядаються колаборативна фільтрація, контент-орієнтовані підходи, гібридні системи рекомендацій, а також сучасні методи ранжування на основі моделей машинного навчання (learning to rank) [41]. Третій напрям - інфраструктурні аспекти: масштабовані архітектури для обробки подій у реальному часі, використання стрімінгових платформ (Kafka), кешування (Redis), та підходи до забезпечення цілості транзакцій при високих навантаженнях [1]. У зарубіжних статтях також

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 11   |

описуються питання безпеки платежів, токенизації платіжних даних і практики інтеграції з платіжними провайдерами [36].

Вітчизняні джерела доповнюють інформацію технічними та нормативними аспектами. У публікаціях українських авторів часто аналізуються архітектурні патерни побудови веб-сервісів, питання вибору СУБД, організації черг повідомлень і забезпечення відмовостійкості [22]. Окремі праці присвячені питанням захисту персональних даних та відповідності локальному законодавству, що є важливим при обробці профілів користувачів і платіжної інформації [49]. Також у вітчизняних джерелах зустрічаються приклади реалізації прототипів сервісів, опис практичних проблем при інтеграції з зовнішніми API і рекомендації щодо тестування та розгортання [50].

Критичний аналіз літератури виявляє кілька прогалин, які обґрунтовують вибір теми бакалаврської роботи. По-перше, у багатьох публікаціях відсутній комплексний підхід, що одночасно поєднує надійну обробку транзакцій, прості механізми персоналізації та легку інтеграцію через стандартизовані API [35]. По-друге, у практичних роботах бракує прикладів реалізації легковагих рекомендаційних механізмів, які можна швидко впровадити в прототипі з обмеженими ресурсами, зберігаючи при цьому можливість подальшого масштабування до ML-рішень [41]. По-третє, питання idempotency та надійного журналювання транзакцій часто розглядаються поверхнево, хоча саме ці аспекти критичні для коректної роботи платіжних сценаріїв у реальному середовищі [35].

Огляд патентної та технічної документації показує, що існують готові рішення для окремих задач: платіжні шлюзи пропонують стандартизовані API і токенизацію, аналітичні платформи - інструменти для збору подій і побудови метрик, а провайдери рекомендацій - готові сервіси з API [36]. Проте інтеграція всіх цих компонентів у єдиний, легкий у розгортанні сервіс, орієнтований на ігрову монетизацію, зустрічається рідше. У патентних описах часто фігурують складні алгоритми ранжування і персоналізації, але їх практична реалізація вимагає значних обчислювальних ресурсів і складної інфраструктури, що не завжди прийнятно для прототипів або стартапів [28].

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 12   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Методологічні підходи, описані в літературі, дають корисні орієнтири для проєктування системи. Рекомендовано застосовувати поділ на шари (API, бізнес-логіка, шар збереження даних), використовувати асинхронні черги для обробки подій і фонових задач, а також впроваджувати idempotency-ключі для фінансових операцій [22]. Для персоналізації пропонується починати з простих правил і популярності, поступово переходячи до статистичних методів і моделей ML при накопиченні даних [41]. У роботах з тестування підкреслюється важливість A/B-тестування для оцінки впливу персоналізованих пропозицій на конверсію і утримання [13].

У підсумку, аналіз літератури показує, що тема поєднання монетизації та персоналізації у вигляді інтелектуального веб-сервісу має як теоретичну базу, так і практичну потребу. Виявлені прогалини - відсутність простих, інтегрованих прототипів і недостатня увага до idempotency та легкості розгортання - обґрунтовують вибір напрямку для бакалаврської роботи і визначають основні завдання для проєктування та реалізації прототипу [35].

## 1.2. Основні поняття та визначення інженерії ПЗ в даному домені

У цьому підпункті зібрано і систематизовано нормативні акти, технічні стандарти та патентні матеріали, що мають прямий вплив на

проєктування й реалізацію веб-сервісу для монетизації та персоналізації ігрового досвіду. Мета - визначити правові обмеження, технічні вимоги та існуючі інноваційні рішення, які слід врахувати при розробці прототипу, а також окреслити ризики.

### Нормативно-правова база

- Захист персональних даних. Обробка профілів користувачів і подій передбачає роботу з персональними даними, тому необхідно дотримуватися вимог національного законодавства щодо захисту персональних даних. Передбачено мінімізацію збережуваних полів, реалізацію механізмів видалення даних за запитом та документування підстав для обробки [49]. Для міжнародних

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 13   |

сценаріїв інтеграції слід враховувати вимоги регіональних актів про захист даних, які впливають на зберігання і передачу інформації між юрисдикціями.

- Платіжні регламенти і стандарти. При роботі з платіжними операціями важливо дотримуватися вимог платіжних систем і стандартів безпеки платіжних даних. Рекомендовано використовувати токенизацію платіжних реквізитів і не зберігати чутливі дані у власних базах [27]. Для інтеграції з платіжними провайдерами слід враховувати їхні правила щодо callback-повідомлень, підписів і верифікації транзакцій [36].

- Ліцензійні та контрактні обмеження. Використання сторонніх бібліотек, SDK платіжних провайдерів або аналітичних сервісів вимагає перевірки ліцензійних умов. У проєкті передбачено застосування відкритих і комерційних компонентів з урахуванням сумісності ліцензій та можливих обмежень на комерційне використання [50].

- Регуляторні вимоги до ігрової індустрії. У випадках, коли монетизація пов'язана з азартними механіками або елементами, що підпадають під регулювання, необхідно враховувати спеціальні норми щодо реклами, вікових обмежень і звітності. На етапі прототипу уникаються механіки, що можуть трактуватися як азартні [26].

### **Технічні стандарти і рекомендації**

- Протоколи і формати обміну. Для API обрано REST з JSON-форматом, що відповідає широким практикам. Потрібно дотримуватися RFC-стандартів для HTTP, коректного використання кодів стану і заголовків, а також застосовувати OpenAPI-специфікацію для документування контрактів [24].

- Автентифікація і авторизація. Для безпечної авторизації застосовано JWT-токени з коротким терміном життя і механізмом оновлення. В планах використовувати стандарти OAuth2 для сценаріїв делегованого доступу та забезпечити захист refresh-токенів [2].

- Безпека веб-додатків. Враховано рекомендації OWASP Top 10: валідація вхідних даних, захист від SQL-ін'єкцій, CSRF, XSS, контроль доступу, безпечне зберігання секретів. Для передачі даних застосовано TLS. Для фінансових

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                          | 14   |

ендпоінтів передбачено rate limiting і механізми виявлення аномалій [25].

- Ідемпотентність. Для запобігання дублюванню транзакцій рекомендовано впровадити idempotency-ключі на рівні API, що дозволяє безпечно повторювати запити при мережових збоїх [35].

- Логування і трасування. Для аудиту фінансових операцій і відстеження помилок передбачено централізоване логування з кореляційними ідентифікаторами, що відповідає практикам для розслідування інцидентів і відновлення стану [9].

#### Патентний ландшафт і інтелектуальна власність

- Патенти в області рекомендаційних систем. У патентних базах присутні описи різних алгоритмів персоналізації, методів ранжування і систем, що комбінують поведінкові сигнали з бізнес-правилами. Багато патентів орієнтовані на масштабні ML-підходи і специфічні способи ранжування, які вимагають значних обчислювальних ресурсів. Для прототипу обрано прості евристичні підходи і правила, що знижує ризик порушення патентних прав, але при комерціалізації слід провести патентний аудит [28].

- Патенти в області платіжних технологій. Існують патентні рішення, що стосуються токенизації, безпечної обробки транзакцій і механізмів верифікації. При інтеграції з платіжними провайдерами важливо використовувати їхні офіційні SDK або API, що зазвичай покривають питання ліцензування і сумісності [36].

- Патентні ризики при використанні сторонніх алгоритмів. Використання комерційних рекомендаційних сервісів або бібліотек може бути обмежено ліцензіями і патентами; тому для прототипу рекомендовано застосовувати відкриті алгоритми і власні прості реалізації, а при переході до промислового рівня - провести юридичну перевірку [50].

#### Аналіз відповідності та рекомендації

- Відповідність правовим вимогам. Для відповідності нормам обробки персональних даних передбачено мінімізацію зберезуваної інформації, механізми видалення і журналювання доступу. Для платіжних операцій -

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 15   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

уникнення зберігання чутливих реквізитів і застосування токенизації [27].

- Відповідність технічним стандартам. Архітектура і протоколи відповідають загальноприйнятим стандартам REST, HTTP, TLS і практикам безпечної розробки. Для транзакцій передбачено використання ACID-гарантій у критичних операціях і асинхронних патернів для аналітики [22].

Проаналізована нормативна, технічна та патентна документація визначає чіткі рамки для розробки прототипу: необхідно забезпечити захист персональних даних, безпечну обробку транзакцій, відповідність стандартам веб-безпеки і уникати потенційних патентних конфліктів. Обрані технічні підходи (реляційна СУБД для транзакцій, кешування, черги для асинхронної обробки, JWT для автентифікації) відповідають цим вимогам і дозволяють побудувати гнучкий прототип, готовий до подальшої еволюції та юридичної перевірки при масштабуванні [7].

### **1.3. Сучасні технології та тенденції , що застосовуються в області сервісних вебсайтів**

Монетизація через продаж косметичних предметів - скіни - стала одним із найпоширеніших і найменш конфліктних способів заробітку в ігровій індустрії [48]. У багатьох проєктах саме косметика дозволяє зберегти ігровий баланс, одночасно даючи гравцям можливість самовираження та демонстрації статусу. У рамках цієї бакалаврської роботи реалізовано саме пряму покупку скінів, тому далі описано ключові підходи й практики, які мають значення для такої моделі, без заглиблення в механіки, що створюють елементи азарту або вимагають складної інфраструктури.

Перш за все, варто відзначити простоту і прозорість моделі прямих покупок: користувач бачить товар, його ціну і одразу отримує бажаний предмет [29]. Такий підхід мінімізує юридичні ризики і підвищує довіру аудиторії, особливо коли відсутні випадкові винагороди. Для підвищення середнього чеку часто застосовуються пакетні пропозиції, тимчасові знижки та набори, що

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 16   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

створюють відчуття вигоди [38]. Також ефективною практикою є введення віртуальної валюти як проміжного шару: продаж пакетів валюти дозволяє гнучкіше формувати ціни і стимулювати більші покупки через бонуси за поповнення [48].

Персоналізація пропозицій відіграє важливу роль у підвищенні конверсії. Навіть прості правила - показати користувачу схожі за стилем шкіни, підкреслити популярні предмети серед гравців його рівня або

запропонувати знижку на предмети, які переглядав раніше - дають помітний ефект [41]. У прототипі застосовано евристичні підходи: базові правила фільтрації за атрибутами профілю. Це дозволяє отримати швидкий приріст конверсії без складних ML-моделей і великих обчислювальних витрат, зберігаючи при цьому можливість еволюції системи до більш просунутих алгоритмів [21].

Технічні аспекти реалізації продажу шкінів мають першочергове значення для коректної роботи сервісу. Необхідна гарантія атомарності транзакцій, журналювання з унікальними ідентифікаторами, механізми idempotency для запобігання дублюванню платежів і чітка обробка callback-повідомлень від платіжних провайдерів [35]. У прототипі передбачено абстрактний шар адаптерів для інтеграції з різними платіжними шлюзами, а також базові правила валідації транзакцій і журналювання для подальшого розслідування спірних операцій [36, 9].

Економічні та регуляторні нюанси теж не можна ігнорувати. Витрати на обробку платежів і комісії провайдерів впливають на кінцеву ціну, а податкові зобов'язання - на чистий дохід [27]. У деяких юрисдикціях механіки, що нагадують азартні ігри (наприклад, лут-бокси), підпадають під суворе регулювання, тому уникнення таких механік знижує ризики [26]. Також важливо передбачити політику повернень і прозорі умови купівлі, що підвищує довіру користувачів і зменшує кількість спірних транзакцій [29].

Модель продажу шкінів як основний канал монетизації в рамках цієї бакалаврської роботи обрана через її простоту, прозорість і відносно низькі

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 17   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

юридичні бар'єри [48]. Поєднання фіксованих цін або пакетних пропозицій з простими персоналізованими рекомендаціями і надійною транзакційною логікою дає практичну і гнучку платформу для подальшого розвитку - від додавання віртуальної валюти до інтеграції більш складних алгоритмів персоналізації та маркетингових інструментів [38]. Персоналізація в іграх перестала бути просто модним словом - це реальний інструмент, який

змінює взаємодію гравця з продуктом і впливає на комерційні показники. У контексті цієї бакалаврської роботи персоналізація розглядається як набір практичних підходів, що дозволяють підвищити релевантність пропозицій (наприклад, скіни), збільшити конверсію та утримання користувачів без необхідності впровадження складних ML-інструментів одразу. Нижче подано огляд підходів, які мають сенс для прототипу та подальшого розвитку платформи.

Початковий рівень персоналізації - це правила й евристики. Простота їхньої реалізації робить їх ідеальними для прототипу: фільтрація за атрибутами профілю (рівень, стиль, попередні покупки), показ популярних предметів у певній когорті користувачів, або рекомендації на основі останніх переглядів. Такі правила дають швидкий ефект без великих витрат на інфраструктуру: наприклад, якщо користувач часто переглядав скіни певного кольору чи тематики, система може підвищити пріоритет схожих пропозицій. У практичному застосуванні це виглядає як набір простих SQL-запитів і кешованих агрегатів, що повертають релевантні елементи за мілісекунди [22].

Другий рівень - статистичні методи і прості моделі ранжування. Тут застосовуються метрики популярності (частота покупок, переглядів), коефіцієнти конверсії для конкретних пропозицій і базові скорингові формули, які комбінують кілька факторів (наприклад, релевантність за профілем  $\times$  популярність  $\times$  свіжість). Такий підхід дозволяє отримати більш тонке ранжування без повноцінного ML-пайплайну: дані агрегуються у фонових задачах (черги, батчі), обчислюються скорингові значення і зберігаються у кеші для швидкого доступу [32]. Для прототипу це оптимальний компроміс між

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 18   |

якістю рекомендацій і простотою реалізації.

Колаборативна фільтрація у спрощеному вигляді теж може бути корисною: не обов'язково запускати матричні факторизації в реальному часі - достатньо побудувати матрицю «користувач-предмет» для активних

користувачів і виконувати періодичні обчислення схожості. Навіть прості підходи на основі спільних покупок (users who bought X also bought Y) дають помітний приріст релевантності. Для прототипу рекомендовано реалізувати такі алгоритми як опціональні модулі, що запускаються у фоновому режимі і оновлюють індекси рекомендацій [41].

Підхід на основі контенту підходить, коли предмети мають багаті атрибути (стиль, рідкість, тематика). У цьому випадку профіль користувача формується як вектор атрибутів, а рекомендації генеруються шляхом пошуку найближчих за векторною відстанню предметів. Така методика добре поєднується з правилами і статистикою: наприклад, якщо користувач віддає перевагу «неоновим» шкінам, система може комбінувати популярність і контентну схожість для формування списку пропозицій [41].

Важливий аспект - збір і обробка подій. Персоналізація працює лише на якісних даних: перегляди, кліки, покупки, час сесії, реакції на акції. Для прототипу передбачено просту подієву модель: події відправляються у чергу, агрегуються у батчі і зберігаються у таблицях або time-series сховищі. Такий підхід дозволяє уникнути навантаження на транзакційні бази і дає можливість поступового ускладнення аналітики - від простих агрегатів до моделей ML [1].

Не можна ігнорувати питання інтерпретованості та контролю. Персоналізація, навіть проста, повинна бути прозорою для бізнесу: правила і ваги скорингових формул мають бути конфігуровані через адмін-інтерфейс або конфіг-файли. Це дає змогу маркетингу швидко тестувати гіпотези і уникати непередбачуваних результатів, які можуть знизити довіру користувачів [21].

Етичні та регуляторні аспекти персоналізації теж важливі. Персоналізовані пропозиції не повинні експлуатувати вразливі групи або створювати агресивні патерни нав'язливих покупок; необхідно враховувати

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 19   |

політики конфіденційності і вимоги до обробки персональних даних [49].

Технічно доцільно будувати персоналізацію як набір модулів: подієвий інжест, агрегатори, скоринговий сервіс і API для отримання рекомендацій. Така модульність дозволяє почати з правил і статистики, а потім підключати більш складні моделі без перебудови архітектури. Для прототипу рекомендовано реалізувати базовий скоринговий сервіс з кешем, фоновими задачами для оновлення індексів і простим інтерфейсом для експериментів [22].

У підсумку, персоналізація в контексті продажу шкінів має бути прагматичною: почати з правил і популярності, додати статистичні дані і просту контентну фільтрацію, паралельно збираючи дані для майбутніх ML-моделей. Такий шлях дозволяє отримати швидкий ефект, мінімізувати витрати і зберегти можливість еволюції системи до більш складних рішень у майбутньому [31].

У роботі над прототипом веб-сервісу особлива увага приділялася тому, як саме рухаються дані всередині системи - від моменту ініціації дії користувача до остаточного оновлення профілю, журналу транзакцій і генерації персоналізованих пропозицій [1]. Описана нижче структура інформаційних потоків відображає практичний підхід до організації взаємодії компонентів, що дозволяє забезпечити надійність, масштабованість і простоту відлагодження [22].

Початковий потік - потік подій користувача. Кожна взаємодія з інтерфейсом (перегляд предмета, клік на пропозицію, початок покупки, завершення оплати) фіксується як подія з мінімальним набором атрибутів: ідентифікатор користувача, тип події, ідентифікатор об'єкта (наприклад, `skin_id`), часовий штамп, контекст (платформа, версія клієнта) і додаткові метрики (час сесії, джерело трафіку) [1]. Події надсилаються на API-ендпоінт збору подій і миттєво ставляться в чергу повідомлень [1]. Використання черги дозволяє відокремити синхронну взаємодію з клієнтом від фонові обробки: клієнт отримує швидку відповідь про

прийняття події, а подальша агрегація і аналіз виконуються асинхронно [1].

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 20   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Другий важливий потік - потік транзакцій. Він починається з ініціації платежу на стороні клієнта і проходить через кілька етапів: створення транзакції у внутрішньому журналі (status = initiated), перенаправлення на платіжний шлюз або виклик платіжного API, отримання callback-повідомлення від провайдера, валідація підпису callback, оновлення статусу транзакції (confirmed/failed) і, у разі успіху, нарахування внутрішнього балансу або видача предмета [36]. Для уникнення дублювання транзакцій застосовується idempotency-ключ: при повторному надсиланні запиту з тим самим ключем система ідентифікує вже оброблену операцію і повертає її результат без повторного нарахування [35]. Важливим елементом є журнал транзакцій з кореляційними ідентифікаторами, що дозволяє відтворити послідовність подій при розслідуванні спірних операцій [5].

Третій потік - потік персоналізації. Події користувача, агреговані у фонових задачах, надходять у підсистему, де обчислюються агрегати [1]. На основі цих агрегатів формуються індекси [22]. Рекомендаційний сервіс запитує ці індекси і повертає ранжований список персоналізацій у відповідь на запит клієнта [41].

Четвертий потік - потік логів і моніторингу. Кожна критична операція (транзакція, помилка валідації, відхилений callback) генерує лог-запис з кореляційним id [9]. Логи централізовано збираються у стек логування і використовуються для аудиту, трасування інцидентів і побудови дашбордів [9]. Наявність кореляційних id дозволяє відстежити шлях конкретного запиту через усі сервіси і швидко локалізувати проблему [9].

Питання безпеки і конфіденційності інтегровані у потоки: чутливі поля не передаються у логах, токени і секрети зберігаються у секрет-менеджері, а доступ до адміністративних ендпоінтів обмежено

ролями і аудитом [25]. Для відповідності нормативним вимогам передбачено механізм видалення персональних даних за запитом і анонімізацію агрегатів для аналітики [49]. На рисунку 1.1 зображена стрілкова діаграма, що показує потоки подій від користувача до аналітики, платежів і модерації відгуків.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 21   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

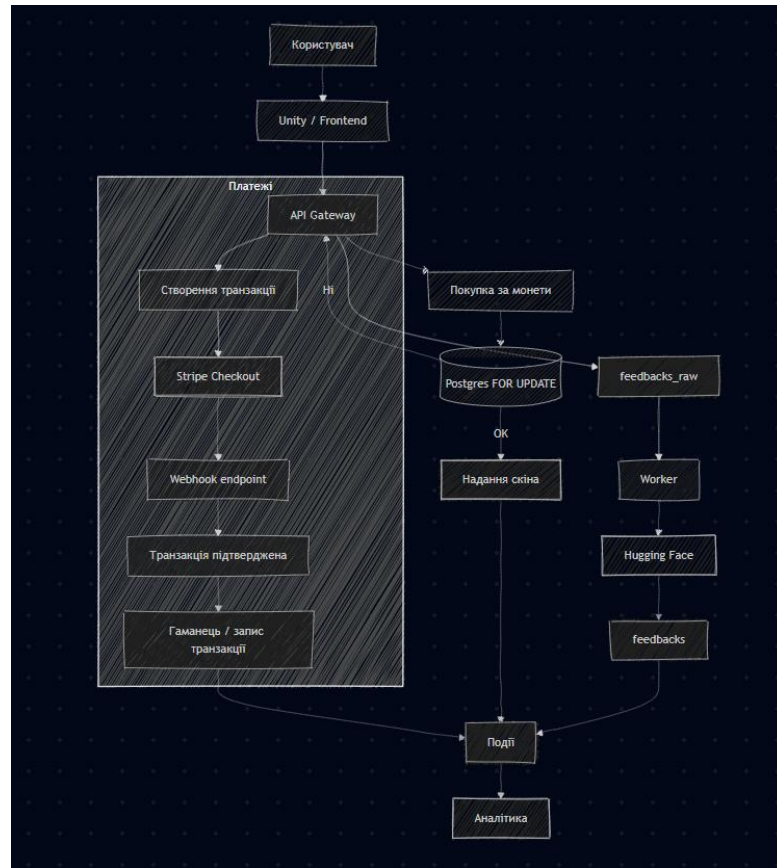


Рисунок 1.1 - Інформаційні потоки між компонентами системи

На практиці така структура інформаційних потоків забезпечує гнучкість: нові джерела подій або додаткові платіжні провайдери підключаються через стандартизовані адаптери [24], а рекомендаційні алгоритми можуть еволюціонувати від простих правил до ML-моделей без зміни базової логіки потоків [22].

#### 1.4. Висновки по розділу

У процесі створення прототипу інтелектуального веб-сервісу для монетизації та персоналізації виявлено низку аспектів, які потребують подальшої опрацювання і становлять логічну дорожню карту для розвитку проєкту. Ці

питання не є критичними для базової функціональності, але їх вирішення значно підвищить надійність, комерційну привабливість і готовність платформи до інтеграції з ігровими клієнтами та зовнішніми сервісами.

Перший блок невирішених питань стосується повноти автоматизованого тестування. Наявні юніт-тести покривають ключові бізнес-правила, а інтеграційні тести перевіряють базову взаємодію з БД і основними ендпоінтами. Проте відсутні комплексні сценарії навантажувального тестування, тривалі стрес-тести та автоматизовані регресійні набори для перевірки поведінки системи при оновленнях [13]. Розширення тестового покриття передбачає: створення сценаріїв імітації пікових навантажень на транзакційні ендпоінти [18], перевірку коректності обробки idempotency-ключів при повторних запитах [35], тестування механізмів компенсації (saga) у розподілених процесах [28], а також автоматизацію А/В-експериментів для оцінки впливу персоналізації на конверсію [31]. Наявність CI-конвеєра, що запускає повний набір тестів при кожному коміті, дозволить швидко виявляти регресії і підвищить довіру до релізів [14].

Другий напрям - впровадження системи щоденних бонусів і механік утримання. Простий модуль, який нараховує внутрішню валюту за щоденний вхід, може значно підвищити активність користувачів і створити

основу для подальших маркетингових кампаній. Для коректної реалізації потрібні правила нарахування (послідовні дні, мультиплікатори за серії входів), захист від шахрайських схем (наприклад, фіксація IP-змін і валідація часових зон) [11], а також механізм відображення бонусів у профілі користувача. Інтеграція бонусної логіки з рекомендаційним модулем дозволить пропонувати релевантні витрати бонусів на скіни, що підвищить середній чек і конверсію. Важливо також передбачити аналітику ефективності бонусів: відстеження retention, частоти повернень і впливу на LTV [4].

Третє питання - готовність до інтеграції з ігровим клієнтом, зокрема Unity. На поточному етапі гра перебуває в розробці іншою командою, тому інтеграція відкладена до моменту узгодження API-контрактів і форматів подій. Для

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 23   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

мінімізації ризиків слід підготувати набір чітко документованих контрактів (OpenAPI) [24], приклади запитів/відповідей, SDK або клієнтські бібліотеки для популярних мов [36], а також тестовий стенд з моками платіжних провайдерів і симуляторами клієнтських подій [10]. Важливо узгодити формат подій, механізми кореляції сесій і підходи до автентифікації (наприклад, OAuth2 або JWT) [2], щоб інтеграція не вимагала значних змін у клієнтській частині. Оскільки робота над грою ведеться паралельно, необхідно встановити канали комунікації з розробником клієнта для синхронізації версій API і тестування інтеграційних сценаріїв у реальному середовищі [50].

Четвертий напрям пов'язаний із розширенням платіжної інфраструктури. Прототип використовує абстрактний шар адаптерів для платіжних шлюзів, але для промислового застосування потрібно підключити реальних провайдерів, налаштувати токенизацію платіжних даних [27], обробку чарджбеків і механізми звітності [26]. Також важливо врахувати комісії провайдерів і податкові зобов'язання при формуванні цін. Для зниження ризиків рекомендується інтегруватися з провайдерами, що надають готові SDK і відповідають стандартам безпеки (PCI-DSS) [36], або використовувати посередницькі сервіси, які беруть на себе зберігання чутливих даних [50].

П'ятий аспект - виявлення шахрайства і захист від фроду. На прототипі реалізовано базові правила валідації транзакцій, але для реального запуску потрібні механізми виявлення аномалій: скорингові моделі транзакцій, правила виявлення підозрілих патернів (швидкі повторні покупки, підозрілі IP-адреси, незвичні комбінації пристроїв) [11], інтеграція з сервісами верифікації платіжних даних і можливість автоматичного блокування підозрілих операцій з подальшим ручним розглядом [26]. Впровадження таких механізмів підвищить безпеку платформи і зменшить фінансові ризики.

Шосте питання - еволюція рекомендаційного модуля. Наразі застосовано евристичні правила і прості скорингові формули, що дають швидкий ефект при обмежених даних. Подальший розвиток передбачає збір більшого обсягу подієвих даних, побудову ML-пайплайнів, впровадження колаборативної

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 24   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

фільтрації або моделей ранжування (learning to rank) [41]. Для цього потрібно організувати збереження навчальних датасетів, інструменти для тренування моделей, систему версіонування моделей і механізм безпечного розгортання (canary releases, A/B-тести) [6, 31]. Важливо також забезпечити інтерпретованість моделей і можливість швидкого відключення при виявленні негативного впливу на бізнес-метрики [21].

Сьоме питання - аналітика і метрики. Для прийняття обґрунтованих рішень потрібні дашборди з ключовими показниками: конверсія пропозицій, ARPU, ARPPU, LTV, retention по когортам, середній чек, ефективність промо-кампаній. Наразі зібрані базові метрики, але необхідно налаштувати регулярні звіти, автоматичні алерти при відхиленнях [19] і можливість сегментації користувачів для таргетованих кампаній. Це дозволить оперативно реагувати на зміни в поведінці аудиторії і коригувати правила персоналізації [4].

Восьме - питання юридичної відповідності і політик конфіденційності. Для масштабного запуску потрібно підготувати політику обробки персональних даних, механізми отримання згоди, процедури видалення даних за запитом і відповідність локальному законодавству [49]. Якщо планується робота з міжнародною аудиторією, слід врахувати регламенти інших юрисдикцій щодо передачі даних [27].

Нарешті, організаційні аспекти інтеграції з командою розробки гри. Оскільки клієнтська частина ще в стадії розробки іншим розробником, необхідно погодити терміни, формат тестових середовищ і механізм обміну артефактами. Рекомендовано встановити регулярні синхронізації, спільний репозиторій з контрактами API і тестовий стенд для інтеграційних перевірок [24, 14]. Це дозволить уникнути затримок і забезпечить плавний перехід від прототипу до реальної інтеграції.

Пріоритети подальшої роботи: розширення автоматизованого тестування і CI [13, 14], впровадження механіки щоденних бонусів [11], підготовка та документування API для інтеграції з клієнтом [24], підключення платіжних провайдерів і посилення фрод-контролю [36, 26], еволюція рекомендаційного

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 25   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

модуля і налаштування аналітичних дашбордів [41, 4]. Виконання цих кроків зробить платформу більш готовою до комерційного використання і дозволить плавно масштабувати функціонал у відповідь на потреби користувачів і бізнесу.

Отримавши та навчившись на певному потоці інформації, узагальнюємо, що подальша робота залишається об'ємною, але вже має право на існування (та навіть на певну ланку конкуренції).

Кожна зміна в проєкті робить його ближче до комерційного релізу і професійної перспективи.

У разі збільшення потоку даних та оновлень, є потреба в зміні середовища розгортання та розширенні команди розробників у випадку значних дедлайнів та перешкод в процесі розгортання чи розробки.

Варто підкреслити, що успішність подальшого розвитку платформи залежатиме не лише від технічних рішень, а й від організації процесів управління проєктом.

Додатково до вже зазначених напрямів, варто сформувати поетапну дорожню карту технічних робіт з чіткими критеріями готовності для кожного етапу: завершення повноцінних тривалих стрес-тестів і верифікація SLA для транзакційних ендпоінтів, впровадження CI/CD з канарними релізами і автоматизованими регресійними тестами, а також підключення централізованого моніторингу й алертингу для ключових метрик (latency, error\_rate, RPS) [13, 14, 19].

Не менш важливим є посилення платіжної та фрод-інфраструктури: інтеграція з реальними провайдерами, налаштування токенизації і обробки чарджбеків.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 26   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

## РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

### 2.1. Аналіз вимог до веб-сервісу монетизації та персоналізації

Аналіз вимог до веб-сервісу починається з визначення його основної ролі: забезпечити надійну обробку транзакцій при купівлі косметичних предметів (скіни), збір і агрегацію відгуків користувачів та надавання хорошого API для подальшої інтеграції з клієнтськими застосунками [24]. Вимоги формулюються з урахуванням пріоритетів прототипу - коректність фінансових операцій і базова персоналізація та можливості еволюції системи до більш складних сценаріїв [22].

Функціональні вимоги охоплюють набір базових сервісів: реєстрація та автентифікація користувачів з підтримкою токенів; управління профілями та балансами; створення, підтвердження та журналювання транзакцій; механізми обробки callback-повідомлень від платіжних провайдерів; збір подій (перегляди, кліки, покупки) з подальшою агрегацією; адміністративні інтерфейси для управління товарами [2]. Для кожної функції визначено очікувану поведінку: наприклад, транзакція має проходити через стани «ініційовано - підтверджено - відхилено - відкат», а для фінансових ендпоінтів передбачено обробку idempotency-ключів, щоб уникнути дублювання нарахувань при повторних викликах [35].

Нефункціональні вимоги визначаються критеріями якості сервісу. Продуктивність: відповіді на операції читання профілю і запити рекомендацій повинні укладатися в межі сотень мілісекунд при типовому навантаженні прототипу; критичні транзакційні операції - у межах прийнятної латентності для користувача [13]. Масштабованість: архітектура має дозволяти горизонтальне масштабування обробників запитів і компонентів аналітики, використовуючи кешування і черги для згладжування піків [22]. Відмовостійкість: забезпечення атомарності операцій у межах транзакційної бази даних, реплікація даних і механізми компенсації для розподілених процесів (патерни saga) [28]. Безпека і конфіденційність: шифрування трафіку (TLS), захист секретів у

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 27   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

секрет-менеджері, мінімізація зберезуваних персональних полів, обмеження доступу до адміністративних функцій і аудит дій відповідно до рекомендацій OWASP [25].

Особлива увага приділяється вимогам до API. Взаємодія має бути реалізована через REST-контракти з чітко визначеними JSON-схемами запитів і відповідей, версіонуванням у шляху (наприклад, /api/v1/) та уніфікованим форматом помилок, що містить код, повідомлення і кореляційний ідентифікатор для трасування [24]. Для фінансових ендпоінтів обов'язкові механізми idempotency-ключів і коректна обробка callback-повідомлень від платіжних провайдерів з валідацією підписів [36]. Аутентифікація - через JWT з коротким терміном життя і механізмом оновлення (refresh-токени) [2].

Вимоги до збору та обробки подій визначаються моделлю подієвого інжесту: події приймаються синхронно, ставляться в чергу повідомлень і обробляються асинхронно для агрегації та оновлення індексів рекомендацій (event streaming, Kafka-подібні підходи) [1].

Формат подій стандартизований: user\_id, event\_type, object\_id, timestamp, context [1]. Черги дозволяють відокремити критичні транзакційні шляхи від аналітичних задач і забезпечити стійкість до пікових навантажень [22]. Результати агрегацій зберігаються у швидкодоступних індексах і кешуються (Redis-патерни) для зниження латентності при запитах рекомендацій [32].

Безпека платіжних операцій і відповідність нормативам - окремий блок вимог. Чутливі платіжні дані не зберігаються у відкритому вигляді; інтеграція з платіжними провайдерами має відбуватися через токенозовані механізми або офіційні SDK провайдерів, що відповідають стандартам безпеки (PCI-DSS) [27]. Журнал транзакцій повинен містити повну історію з кореляційними id для аудиту і розслідування чарджбеків [5].

Критерії прийнятності реалізації прототипу включають: коректну обробку транзакцій з idempotency, наявність CRUD-інтерфейсів для товарів і профілів, збір подій з черг і базова генерація рекомендацій за правилами і популярністю [35]. Пріоритети реалізації визначено так: спочатку - надійний

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 28   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

облік транзакцій і журналювання; далі - автентифікація і управління профілями; потім - збір подій і простий модуль обробки відгуків; паралельно - логування і моніторинг (Sentry/ELK-практики) [34, 9].

Аналіз ризиків виявляє кілька важливих обмежень. Ризик дублювання транзакцій при відсутності idempotency, ризик шахрайства при недостатній валідації платежів, залежність від зовнішніх платіжних провайдерів і юридичні обмеження щодо обробки персональних даних [35, 11, 49]. Таблична візуалізація ключових функціональних і нефункціональних вимог зображена на рисунку 2.1.

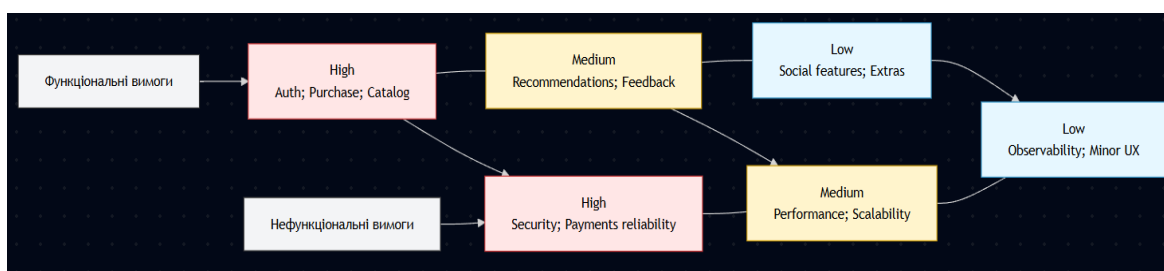


Рисунок 2.1 - Матриця пріоритетів вимог

Підсумкова архітектурна вимога - модульність: сервіси повинні бути розбиті на чіткі компоненти (автентифікація, транзакції, подієвий інжест, адміністрація), кожен з яких має власний API і може масштабуватися незалежно [22].

Такий підхід забезпечує швидку ітерацію, полегшує тестування (CI/CD з GitHub Actions, контейнеризація через Docker/Compose) і дозволяє поступово додавати функціонал - наприклад, механіку щоденних бонусів або інтеграцію з клієнтом на Unity - без радикальної перебудови системи [14, 7, 48].

## 2.2. Формулювання функціональних та нефункціональних вимог

Початковий принцип - починати з простого і робити кроки до складнішого. На старті реалізовано набір правил і евристик, що дають швидкий ефект при мінімальних витратах ресурсів [1, 22]. Правила включають фільтрацію за атрибутами профілю (рівень гравця, попередні покупки, уподобання за

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 29   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

стилем), показ популярних предметів у когорті та підвищення пріоритету недавно переглянутих позицій [1, 32]. Такий набір дозволяє отримати перші метрики конверсії і зрозуміти, які сегменти користувачів реагують краще [31]. Практична реалізація - прості SQL-запити і кешовані агрегати, що повертають релевантні списки за мілісекунди [32, 24].

Другий шар персоналізації - скорингові формули. Для ранжування пропозицій застосовується комбінований скоринг, який поєднує кілька факторів: релевантність до профілю, популярність серед схожих користувачів, свіжість (новизна предмета) і бізнес-метрики (маржинальність або пріоритет маркетингу) [31, 32]. Кожен фактор нормалізується і отримує вагу, що конфігурується через адмін-інтерфейс [31]. Такий підхід дає гнучкість: маркетинг може швидко змінити ваги, а технічна команда - відстежити вплив змін на KPI [31]. Технічно скоринг обчислюється у фонових батчах, результати зберігаються у Redis і повертаються API для швидкого відображення в клієнті [32, 1].

Колаборативна фільтрація реалізується у спрощеному вигляді: побудова матриці «користувач-предмет» на основі покупок і переглядів, обчислення схожості між предметами через косинусну схожість або Jaccard, періодичне оновлення індексів [41, 1]. Для прототипу достатньо щоденних батчів, які оновлюють індекси рекомендацій [1, 31]. Це дає помітний приріст релевантності без потреби у реальному часі ML-інференсу. Результати комбінуються з правилом популярності і скорингом, щоб уникнути надмірної персоналізації і забезпечити різноманітність пропозицій [31, 32].

Контентна фільтрація застосовується, коли предмети мають багаті атрибути: стиль, колір, тематика, рідкість [41]. Профіль користувача формується як вектор атрибутів на основі історії переглядів і покупок, а рекомендації генеруються через пошук найближчих векторів [41]. Для невеликого каталогу це працює швидко і дає інтуїтивно зрозумілі результати [41]. При зростанні каталогу планується використання індексів для пошуку найближчих сусідів (ANN), що дозволить зберегти швидкість відповіді [41, 32].

Контекстна персоналізація - наступний крок. Враховуються час доби,

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 30   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

платформа, джерело трафіку, поточні акції [31]. Для оптимізації показів застосовуються алгоритми багатозброєних бандитів (contextual bandits), які дозволяють балансувати між вивченням нових пропозицій і використанням відомих кращих варіантів [31, 21]. На практиці це означає, що для частини трафіку пропозиції підбираються адаптивно, а результати експериментів автоматично враховуються у скорингу [31].

Експериментальна платформа - обов'язкова частина персоналізації. Розподіл користувачів по групах, збір метрик (конверсія, середній чек, retention) і статистичний аналіз дозволяють приймати обґрунтовані рішення [31]. Для прототипу реалізовано простий механізм А/В-тестів: випадковий розподіл, фіксація експозиції і збір ключових КРІ [31]. Результати виводяться у дашборди, що дає змогу швидко відкидати неефективні гіпотези [4, 31].

Інтерпретованість і контроль - важливі елементи. Ваги скорингових формул і правила фільтрації зберігаються у конфігураційному сховищі і доступні через адмін-панель [21, 31]. Це дозволяє маркетингу і продакт-менеджерам розуміти, чому конкретний користувач бачить ту чи іншу пропозицію, і оперативно коригувати стратегію [21]. Для моделей ML передбачено логування фіч і прогнозів, щоб у разі негативного впливу можна було швидко відкотити зміни [21, 34].

Збір даних - фундамент персоналізації. Події (перегляди, кліки, покупки, час сесії) фіксуються з мінімальним набором атрибутів: user\_id, event\_type, object\_id, timestamp, context [1]. Події надсилаються на REST-ендпоінт і ставляться в чергу повідомлень; воркери асинхронно агрегують дані і оновлюють індекси [1, 22]. Така архітектура дозволяє уникнути навантаження на транзакційну базу і забезпечує масштабованість [22, 13].

Питання приватності і етики інтегровані у процес: мінімізація збережуваних персональних полів, можливість видалення профілю за запитом, анонімізація агрегатів для аналітики [49]. Персоналізація не повинна експлуатувати вразливі групи або створювати агресивні патерни нав'язливих покупок; обмеження частоти пропозицій і можливість відключення

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                         | 31   |

персоналізації для користувача реалізовані на рівні сервісу [49, 25].

Фрод-контроль - невід’ємна частина системи. Для транзакцій застосовуються idempotency-ключі, валідація підписів callback-повідомлень від платіжних провайдерів і базові скорингові правила для виявлення аномалій (швидкі повторні покупки, підозрілі IP, невідповідність пристроїв) [35, 36, 11]. Підозрілі операції маркуються для ручної перевірки або автоматично блокуються за заданими порогами [26].

Інтеграція з клієнтом Unity потребує узгодження контрактів API і прикладів SDK. Для полегшення підключення підготовлено OpenAPI-специфікацію і приклади запитів, а також план створення простих клієнтських бібліотек [24, 48]. Особлива увага приділяється кореляції сесій і безпечній автентифікації (JWT або OAuth2), щоб клієнт міг коректно передавати події і отримувати персоналізовані пропозиції в реальному часі [2, 24].

Такий поетапний підхід дозволяє отримувати прибуток уже на ранніх етапах, одночасно готуючи платформу до більш складних аналітичних задач і масштабування [38, 31]. План подальшого розвитку включає поступовий перехід від правил і скорингу до ML-моделей, впровадження контекстних bandit-алгоритмів для персоналізації в реальному часі, розширення фрод-контролю і інтеграцію з платіжними провайдерами для повноцінної обробки платежів [41, 36]. Такий поетапний підхід дозволяє отримувати прибуток уже на ранніх етапах, одночасно готуючи платформу до масштабування і більш складних аналітичних задач [31, 6].

Документація і інструменти для розробника включають специфікацію, README з інструкціями зі збірки і запуску, приклади запитів до API і конфігураційні шаблони для локального запуску через Docker Compose [7]. Для полегшення інтеграції з клієнтом підготовлено приклади запитів і рекомендації щодо автентифікації, а також план створення простих SDK для Unity у майбутньому [48].

Нефункціональні вимоги сформульовано так, щоб сервіс працював стабільно під навантаженням, був безпечним і зручним для подальшого розвитку

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 32   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

[22][31]. Відповідь на запити профілю та рекомендацій має бути швидкою - у межах 200–300 мс у типовому режимі, а критичні транзакційні операції повинні укладатися в 500–800 мс; для досягнення цих показників передбачено індексацію в PostgreSQL [13, 32]. Навантажувальні тести повинні підтвердити роботу при очікуваних піках, наприклад 200–500 RPS для API-шару прототипу, з можливістю горизонтального масштабування [13, 18].

Архітектура має підтримувати масштабування окремих компонентів незалежно один від одного, тому сервіси розбиті на модулі: обробка транзакцій, інжест подій, рекомендації, адміністрація [22, 1].

Надійність і доступність реалізуються через реплікацію бази даних, регулярні бекапи і розгортання критичних сервісів у кількох інстансах з балансувальником [22, 16]. Цільовий рівень доступності для прототипу встановлено на 99.5% у робочі години, з планом підвищення до 99.9% при промисловому запуску [6, 17]. Механізми failover і health-checks автоматично виводять з ладу проблемні інстанси і перенаправляють трафік, а для розподілених операцій застосовується поєднання ACID-транзакцій і патерну саги з чіткими компенсаційними діями [28, 22].

Безпека і конфіденційність реалізовані на кількох рівнях: весь трафік шифрується TLS, автентифікація відбувається через JWT з коротким терміном життя і механізмом refresh-токенів [2, 25], секрети зберігаються у секрет-менеджері з ротацією ключів [49]. Чутливі платіжні дані не зберігаються у відкритому вигляді, використовується токенизація через платіжних провайдерів або сертифіковані сервіси, а доступ до адміністративних ендпоінтів обмежено RBAC і аудитом дій [36, 27, 5].

Моніторинг і трасування налаштовані централізовано: метрики продуктивності і бізнес-метрики збираються і візуалізуються [19, 30]; логи централізовано надходять у стек ELK або сумісний сервіс [9, 34]. Кореляційні id у логах дозволяють відстежити шлях запиту через усі сервіси, а алерти сповіщають про відхилення SLA - зростання latency, збільшення частки failed транзакцій або падіння throughput [19, 30]. Distributed tracing допомагає швидко

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 33   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

локалізувати вузькі місця [30, 1].

Тестованість і автоматизація включають CI/CD-конвеєр, який запускає юніт-тести, інтеграційні тести з тестовою БД, статичний аналіз коду і набори навантажувальних тестів перед релізом у тестове середовище [14, 31, 13]. Автоматичні регресійні тести для перевірки idempotency, обробки callback-повідомлень і компенсаційних сценаріїв є обов'язковими, а staged rollout і canary-релізи дозволяють вводити зміни без ризику для основного трафіку [6, 17, 28].

Юридичні та ліцензійні обмеження враховано при виборі сторонніх бібліотек і платіжних провайдерів: перевіряється сумісність ліцензій, обираються провайдери з сертифікатами безпеки, а політики зберігання даних і терміни ретенції документуються і реалізуються на рівні БД [27, 49]. Обмеження ресурсів і квоти реалізовано через rate limiting на рівні користувача і API-ключа, з жорсткішими лімітами для платіжних ендпоінтів, щоб захистити систему від зловживань [25, 36].

Завдання на найближчий період - підтвердити показники продуктивності навантажувальними тестами, налаштувати повний CI/CD-потік з автоматичними регресіями, відпрацювати процедури відновлення після аварій і завершити впровадження моніторингу з алертами для ключових SLA [13, 14, 19]. Це дозволить отримати стабільну платформу, готову до інтеграції з клієнтом і подальшого масштабування без радикальної перебудови архітектури [22, 16].

### 2.3. Постановка задачі проектування системи

Статистичні дані та тенденції індустрії ігор подаються тут у форматі практичного огляду, який допомагає зрозуміти, куди рухається ринок і які можливості відкриваються для сервісу монетизації косметичних предметів [38, 29]. Підхід орієнтований на застосування у прототипі: які метрики відстежувати, які механіки працюють найкраще, які ризики варто врахувати при масштабуванні [31]. Інформація подається з акцентом на практичну користь для розробника і

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 34   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

маркетолога, а не як академічний огляд [31].

Глобальна картина ринку залишається сприятливою для проєктів, що пропонують косметичні предмети і мікротранзакції [38, 29]. Ринок ігор демонструє стабільне зростання доходів і кількості активних користувачів, при цьому мобільні платформи утримують лідерство за обсягом виручки, а консолі і ПК зберігають високу середню виручку на користувача [38, 15]. Для продукту, що продає скіни, це означає наявність великої потенційної аудиторії і можливість вибору цільових платформ залежно від стратегії: мобільний фокус для швидкого масштабування аудиторії або кросплатформені рішення для підвищення середнього чеку [48, 29]. Важливо орієнтуватися не лише на загальні цифри, а на поведінкові патерни користувачів у конкретних сегментах - наприклад, гравці мобільних free-to-play проєктів частіше реагують на пакетні пропозиції і тимчасові акції, тоді як аудиторія ПК/консоль більш чутлива до унікальності і рідкості предметів [38, 15].

Монетизаційні тренди показують, що косметика і внутрішня валюта залишаються основними драйверами доходу без руйнування ігрового балансу [38, 29]. Популярність сезонних пропозицій, battle pass-систем і пакетів з бонусами зростає, оскільки ці механіки поєднують елементи утримання і стимулювання витрат [38, 15]. Для прототипу це означає пріоритет у впровадженні внутрішньої валюти, пакетів різного рівня і механік серійних бонусів (наприклад, мультиплікатори при купівлі більших пакетів) [38, 29]. Показовим є ефект price anchoring: надання трьох опцій з різними опорними цінами значно підвищує вибір середнього або більшого пакета, що прямо впливає на ARPU [29, 38].

Live-ops підхід став стандартом для успішних проєктів: регулярні події, щоденні бонуси, тимчасові набори і мікро-кампанії підвищують утримання і стимулюють повторні покупки [15, 38]. Для прототипу варто впровадити простий механізм щоденних подарунків внутрішньої валюти, систему серійних нагород за послідовні входи і обмежені за часом пропозиції [15]. Ці механіки не потребують складних ML-моделей, але

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 35   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

дають помітний приріст retention і частоти транзакцій [31, 15]. Важливо також передбачити аналітику ефективності кожної акції: відстеження cohort retention, частоти повторних покупок і впливу на LTV [31, 4].

Персоналізація пропозицій залишається одним із найефективніших інструментів підвищення конверсії [31, 22]. На старті достатньо правил і скорингових формул, що комбінують релевантність за профілем, популярність і свіжість [32, 31]. Поступове підключення колаборативної фільтрації і простих моделей propensity дозволяє підвищити релевантність без великих інвестицій у ML-інфраструктуру [41, 21]. Для практичної реалізації важливо забезпечити якісний інжест подій (перегляди, кліки, покупки), зберігання агрегатів і можливість швидкого оновлення індексів рекомендацій [1, 22]. Паралельно слід запускати A/B-експерименти для перевірки впливу персоналізації на KPI [31, 6].

## 2.4. Висновки до розділу

Технологічні тренди вказують на необхідність модульної архітектури: розділення транзакційного шару, аналітики і рекомендаційного сервісу дозволяє масштабувати кожен компонент незалежно [22, 16]. Використання брокерів повідомлень для подієвого інжесту, кешів для зниження латентності і окремих індексів для рекомендацій - стандартні практики, які вже довели свою ефективність у індустрії [1, 32, 22].

Для інтеграції з клієнтом Unity корисно підготувати OpenAPI-контракти і прості SDK, що зменшить час на підключення і тестування [24, 48].

Регуляторні та етичні ризики потребують уваги при проєктуванні монетизаційних механік [27, 26]. Механіки, що нагадують азартні ігри (наприклад, лут-бокси), підлягають підвищеній увазі регуляторів у багатьох юрисдикціях, тому варто уникати непрозорих механік або забезпечити повну прозорість умов і ймовірностей [27, 26].

Захист неповнолітніх, політика повернень і обробки чарджбеків, а також можливість видалення персональних даних - обов'язкові елементи для

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 36   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

довгострокової роботи на ринку [49, 26].

У підсумку, ринок дає можливість швидкого старту з простими механіками монетизації і поступового переходу до більш складних персоналізованих рішень [38, 31].

Додатково до технічних пріоритетів варто виділити розвиток інфраструктури спостереження та автоматизації: впровадження observability (логування, метрики, трасування), налаштування CI/CD-конвеєра з автоматичним деплоєм і rollback-механізмами, а також використання хмарних патернів для авто-масштабування сервісів і оптимізації витрат. Це дозволить оперативно виявляти вузькі місця в продуктивності, контролювати латентність транзакцій і забезпечити безперервність бізнес-процесів під час пікових навантажень. Окрему увагу слід приділити кешуванню гарячих даних і оптимізації запитів до бази, щоб знизити навантаження на транзакційний шар без втрати консистентності.

Додатково до технічних і організаційних пріоритетів варто виділити план поступового впровадження observability: поетапне підключення централізованого логуювання, метрик і трасування, налаштування правил оповіщення та дашбордів для оперативного моніторингу SLA транзакційних ендпоінтів і поведінкових метрик (конверсія, retention, LTV). [9, 19] Це дозволить швидко виявляти деградацію сервісу під час пікових навантажень, корелювати інциденти з зовнішніми викликами (webhook-и, HF) і автоматизувати реакцію (скрипти відкату, автоскейлінг) для зниження часу простою. [36, 40, 16]

Управлінський аспект розвитку повинен включати формалізацію KPI і критеріїв переходу між етапами (прототип, пілот, бета, реліз).

## РОЗДІЛ 3. ПРОЕКТУВАННЯ СИСТЕМИ

### 3.1. Розробка архітектури програмного забезпечення

Архітектура спроектована так, щоб забезпечити надійність транзакцій, гнучкість персоналізації і простоту інтеграції з клієнтом на Unity [22, 48].

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 37   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Основні блоки системи - API-шар, сервіс автентифікації, сервіс транзакцій, інжест подій, рекомендаційний сервіс, адміністративний модуль, сховище даних і інфраструктурні сервіси [22, 1]. Кожен блок має чітко визначену відповідальність і взаємодіє з іншими через стандартизовані контракти [24].

API-шар виконує роль фасаду для клієнтських запитів і відповідає за базову валідацію, аутентифікацію через JWT, призначення `correlation_id` і маршрутизацію запитів до відповідних сервісів [2, 25]. Запити, пов'язані з фінансовими операціями, проходять додаткову перевірку `idempotency`-ключів на рівні API, щоб уникнути дублювання транзакцій при повторних викликах [35].

Сервіс автентифікації відповідає за видачу і верифікацію JWT, управління `refresh`-токенами і базовий RBAC для адміністративних ендпоінтів [2, 25]. Токени мають короткий термін життя, а механізм оновлення дозволяє зменшити ризик компрометації сесій [2]. Для підвищення безпеки секрети зберігаються у секрет-менеджері, а всі внутрішні і зовнішні з'єднання шифруються TLS [49, 25].

Сервіс транзакцій реалізує бізнес-логіку купівлі скінів і роботи з внутрішньою валютою. Послідовність операцій: створення транзакції у статусі `initiated`, виклик платіжного провайдера через адаптер, очікування `callback`, валідація підпису `callback` і оновлення статусу транзакції [36, 35]. Для забезпечення атомарності і відкатів застосовується патерн саги: у разі помилки виконуються компенсаційні дії, а інциденти автоматично створюються у системі підтримки з усіма кореляційними даними [28, 5]. Журнал транзакцій зберігає повну історію станів і метадані для аудиту [5].

Адміністративний модуль забезпечує CRUD для товарів. Зміни конфігурації зберігаються у конфігураційному сховищі і поширюються на підписані сервіси через механізм подій конфігурації [1, 22]. Для критичних змін передбачено `staged rollout`: спочатку застосування до невеликої частини трафіку, збір метрик і лише після позитивного результату - повне розгортання [6, 17].

Репозиторій на GitHub слугує відправною точкою: у ньому міститься початкова реалізація бекенду, структура проекту і приклади конфігурацій для локального запуску. Рекомендовано розгорнути локальний стек через Docker

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 38   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Compose з PostgreSQL, додати тести для ключових сценаріїв і узгодити OpenAPI-контракти з розробником клієнта [7, 22, 13]. Компонентна діаграма з позначенням контейнерів зображена на рисунку 3.1.

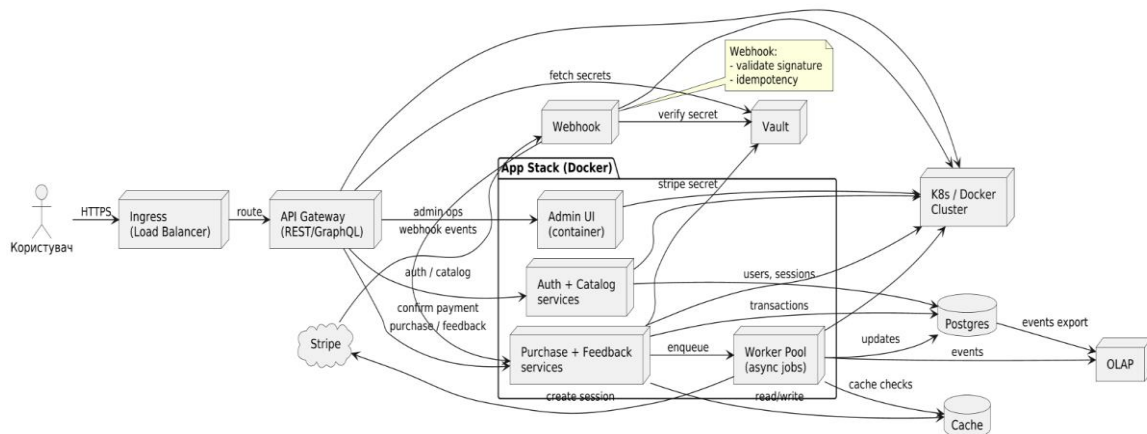


Рисунок 3.1 - Детальна архітектура сервісів і інтеграцій.

### 3.2. Моделювання бізнес-процесів та системних компонентів

Підхід до монетизації в проєкті формувався з урахуванням практичної мети: реалізувати просту, прозору і масштабовану модель продажу косметичних предметів (скіни), яка одночасно дозволяє експериментувати з персоналізацією пропозицій і вимірювати ефективність маркетингових рішень [38, 29]. Нижче подано послідовний опис методів і алгоритмів, які застосовуються на різних рівнях системи - від бізнес-логіки ціноутворення до алгоритмів ранжування пропозицій і механізмів захисту від шахрайства.

Початковий рівень - фіксоване ціноутворення і пакети. Для прототипу обрана модель прямих покупок зі статичними цінами на окремі скіни та пакетні пропозиції (bundle). Такий підхід простий у реалізації, прозорий для користувача і дозволяє швидко отримати перші метрики: конверсію, середній чек, частку платних користувачів [29, 38]. Пакети формуються за правилами: комбінування популярних і менш популярних предметів, надання знижки при купівлі набору, обмежені за часом пропозиції. Технічно це реалізується як набір правил у базі даних товарів і промо-правил, які застосовуються під час формування кошика

[24].

Другий рівень - віртуальна валюта і стимулювання поповнень. Впровадження внутрішньої валюти дозволяє гнучко формувати цінові пропозиції: продаж пакетів валюти зі знижками, бонуси за перше поповнення, мультиплікатори при великих покупках [38]. Алгоритм обробки включає конвертацію реальної валюти у внутрішню, збереження транзакцій у журналі і механізми компенсації при відкатах [5]. Для підвищення середнього чеку застосовується price anchoring: показ кількох пакетів з різними опорними цінами, що стимулює вибір середнього або більшого пакета [29].

Третій рівень - персоналізовані пропозиції і скоринг. Навіть без складних ML-моделей можна значно підвищити релевантність пропозицій за допомогою скорингових формул. Базова формула комбінує релевантність за профілем (сумісність атрибутів скіна з уподобаннями користувача), популярність (частота покупок або переглядів серед схожих користувачів), свіжість (нові предмети отримують бонус), маржинальність (перевага товарів з вищою маржею для розробника) [32, 31]. Кожен фактор нормалізується і зважується, отримане значення використовується для ранжування. Формули конфігуруються через адмін-інтерфейс, що дозволяє маркетингу оперативно змінювати ваги і тестувати гіпотези [31].

Четвертий рівень - колаборативна фільтрація і прості матричні методи. Для підвищення якості рекомендацій застосовується спрощена колаборативна фільтрація: побудова матриці «користувач-предмет» на основі покупок і переглядів, обчислення схожості між предметами або користувачами і періодичне оновлення індексів [41]. Для прототипу достатньо періодичних батч-обчислень (наприклад, щоденних), що зберігають результати у кеші для швидкого доступу [1, 32]. Такий підхід дає помітний приріст релевантності без потреби у реальному часі ML-інференсу.

П'ятий рівень - контентна фільтрація. Якщо скіни мають багаті атрибути (стиль, колір, тематика), профіль користувача можна представити як вектор атрибутів, а рекомендації генерувати через пошук найближчих векторів. Для

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 40   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

невеликого набору предметів це ефективно і швидко; при зростанні каталогу можливе використання індексів для пошуку найближчих сусідів (ANN) [41].

Шостий рівень - контекстні та адаптивні алгоритми (bandits і uplift). Для оптимізації пропозицій у реальному часі застосовуються алгоритми багатозброєних бандитів (contextual bandits), які дозволяють балансувати між вивченням нових пропозицій і використанням відомих кращих варіантів. Контекст включає атрибути користувача, час доби, джерело трафіку. Uplift-моделі допомагають оцінити додатковий ефект від конкретної пропозиції на поведінку користувача, що корисно при плануванні промо-кампаній [31, 21].

Сьомий рівень - A/B-тестування і експериментальна платформа. Для оцінки впливу змін у ціноутворенні або алгоритмах персоналізації необхідна платформа для експериментів: розподіл користувачів по групах, збір ключових метрик (конверсія, ARPU, retention), статистичний аналіз результатів [31]. Експерименти проводяться з урахуванням коректного розподілу і достатньої вибірки, а результати автоматично інтегруються у дашборди [4].

Восьмий рівень - прогнознi моделі LTV і propensity. Для прийняття рішень щодо маркетингових витрат і сегментації корисно прогнозувати lifetime value (LTV) і ймовірність покупки (propensity). Моделі побудовані на історичних даних: частота покупок, середній чек, активність, час від останньої покупки. Навчання моделей відбувається у батч-режимі, результати зберігаються у профілях користувачів і використовуються для таргетування пропозицій [21, 31].

Дев'ятий рівень - фрод-контроль і валідація транзакцій. Для захисту доходів застосовуються скорингові правила транзакцій: перевірка idempotency-ключів, валідація підписів callback-повідомлень від платіжних провайдерів, аналіз аномалій (швидкі повторні покупки, підозрілі IP, невідповідність пристроїв) [35, 36, 11]. Підозрілі операції маркуються для ручної перевірки або автоматично блокуються за заданими порогами. Для складніших сценаріїв передбачено інтегрування зовнішніх сервісів фрод-аналізу [26].

Технічні аспекти реалізації алгоритмів включають розподіл на реальний час і батч-обробку. Критичні транзакційні операції виконуються синхронно з

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 41   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

гарантіями атомарності у PostgreSQL; аналітичні обчислення і тренування моделей - у фонових батчах з використанням черг і воркерів (event streaming, Kafka-подібні підходи) [22, 1]. Результати батчів кешуються у Redis для швидкого доступу [32]. Така архітектура дозволяє зберегти низьку латентність для користувача і масштабувати аналітику незалежно від транзакційного навантаження [13, 18].

Метрики ефективності монетизації - обов'язковий елемент. Відстежуються ARPU, ARPPU, конверсія, середній чек, LTV, retention по когортам, частка платних користувачів [31]. Для кожного експерименту визначається набір KPI і критерії прийнятності змін. Дашборди і алерти дозволяють оперативно реагувати на відхилення [19, 4].

Юридичні та етичні обмеження накладають рамки на вибір методів: уникнення механік, що нагадують азартні ігри (лут-бокси), прозорість умов купівлі, захист неповнолітніх. Конфіденційність даних забезпечується мінімізацією зберезуваних полів і можливістю видалення профілю за запитом [27, 49].

План подальшого розвитку включає поступовий перехід від правил і скорингу до ML-моделей, впровадження контекстних bandit-алгоритмів для персоналізації в реальному часі, розширення фрод-контролю і інтеграцію з платіжними провайдерами для повноцінної обробки платежів [41, 36]. Такий поетапний підхід дозволяє отримувати прибуток уже на ранніх етапах, одночасно готуючи платформу до масштабування і більш складних аналітичних задач [31, 6].

Документація і інструменти для розробника включають специфікацію, README з інструкціями зі збірки і запуску, приклади запитів до API і конфігураційні шаблони для локального запуску через Docker Compose [7]. Для полегшення інтеграції з клієнтом підготовлено приклади запитів і рекомендації щодо автентифікації, а також план створення простих SDK для Unity у майбутньому [48].

Початковий контакт з системою відбувається через клієнтський

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 42   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

застосунок або веб-інтерфейс. Кожен HTTP-запит проходить через шар API-gateway або балансувальник навантаження, де відбувається базова валідація, аутентифікація (JWT) і призначення кореляційного ідентифікатора для подальшого трасування [2, 25, 24]. Для фінансових операцій додатково перевіряється наявність idempotency-ключа; якщо

ключ відсутній - генерується новий і повертається клієнту [35]. Формат запитів і відповідей стандартизовано через OpenAPI, що спрощує узгодження контрактів з клієнтською частиною, зокрема з Unity [24, 48].

Далі запит потрапляє до сервісу обробки транзакцій або сервісу подій залежно від типу дії. Для операцій, пов'язаних із покупкою скіна, послідовність така: створення запису транзакції у PostgreSQL у статусі initiated, відправлення запиту до платіжного провайдера через адаптер, очікування callback-повідомлення [22, 5, 36]. Callback від платіжного провайдера надходить на окремий ендпоінт, де відбувається валідація підпису, перевірка idempotency-ключа і оновлення статусу транзакції (confirmed/failed) [36, 35]. У разі підтвердження - нарахування предмета або внутрішньої валюти виконується в межах тієї ж транзакції БД або через механізм саги з компенсаціями, якщо операції розподілені [28, 5]. Всі зміни логуються з кореляційним id, що дозволяє відтворити повний шлях операції при розслідуванні.

Сервіс відгуків запитує індекси з кешу і, за потреби, звертається до скорингового сервісу для перерахунку ранжування [32, 31]. Якщо кеш не містить потрібного набору, виконується fallback: повернення популярних предметів або застосування простих правил фільтрації за атрибутами профілю [29, 1]. Відповідь рекомендаційного сервісу містить список item\_id з полями score і rationale (коротке пояснення, чому предмет запропоновано), що підвищує прозорість для маркетингу і дозволяє відстежувати вплив змін у скорингу [31, 21].

Адміністративні операції (управління товарами, промо, зміна ваг скорингу) проходять через окремий сервіс конфігурації. Зміни зберігаються у конфігураційному сховищі і поширюються на підписані сервіси через механізм подій конфігурації [1, 22]. Для критичних змін передбачено staged rollout:

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 43   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

спочатку застосування до невеликої частини трафіку, збір метрик, і лише після позитивного результату - повне розгортання [6, 17].

Моніторинг і логування інтегровані у всі потоки. Логи структуровано формуються з полями timestamp, service, level, correlation\_id, user\_id [9, 34]. Метрики продуктивності (RPS, latency, error\_rate) збираються і візуалізуються; налаштовані алерти сповіщають про відхилення SLA, зокрема про зростання часу відповіді на транзакційні ендпоінти або збільшення частки failed транзакцій [19, 30].

Формати обміну стандартизовано JSON-схемами з обов'язковими полями: user\_id, event\_type, object\_id, timestamp, context. Для транзакцій додатково передбачені поля payment\_provider, amount, currency, idempotency\_key, correlation\_id [24, 36, 35].

Системні зв'язки та інформаційні потоки зображені на рисунку 3.2.

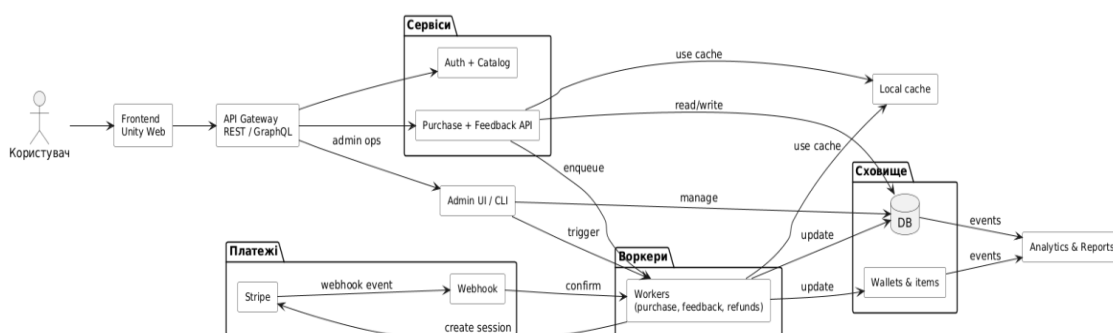


Рисунок 3.2 - Деталізована діаграма компонентів з підсистемами

### 3.3. Вибір та обґрунтування технологій та інструментів розробки

Підхід до вибору технологій будувався на прагненні отримати робочий, зрозумілий і водночас гнучкий прототип, який легко масштабувати й інтегрувати з клієнтськими застосунками [22]. Відразу ставився акцент на стабільності бекенду, надійності збереження фінансових даних і швидкості відповіді для користувача - саме ці критерії визначили остаточний стек технологій і архітектурні рішення [13].

Як основа серверної логіки обрана Java з фреймворком Spring Boot. Такий вибір пояснюється широкою екосистемою, наявністю перевірених бібліотек для побудови REST-сервісів, підтримкою ORM через Spring Data JPA і зручними інструментами для тестування [22]. Spring Boot дозволяє швидко створювати автономні сервіси з мінімальною конфігурацією, що особливо важливо при розробці прототипу з обмеженим часом. Архітектура сервісів спроектована за принципом розділення відповідальностей: контролери обробляють HTTP-запити, сервіси містять бізнес-логіку, репозиторії відповідають за доступ до даних [22].

Для збереження критичних транзакційних даних використано PostgreSQL. Реляційна СУБД обрана через підтримку ACID-гарантій, зручні механізми індексації і транзакційного контролю, що необхідні для коректного обліку балансу та журналу операцій [22]. Структура бази даних спроектована з урахуванням нормалізації для основних сутностей (користувач, профіль, товар, транзакція, подія), а також з додатковими таблицями для агрегатів і метрик, які формуються у фонових задачах [1].

Безпека - окрема складова архітектури. Для автентифікації застосовано JWT з коротким терміном життя і механізмом refresh-токенів [2]; весь трафік шифрується TLS і секрети/ключі розміщуються у секрет-менеджері відповідно до практик безпеки API [25]. Чутливі дані не зберігаються у відкритому вигляді; для захисту від типових веб-вразливостей реалізовано валідацію вхідних даних, захист від SQL-ін'єкцій, обмеження частоти запитів (rate limiting) для критичних ендпоінтів і базовий фрод-скринінг для транзакцій [25, 11]. Журнал транзакцій містить кореляційні id і повну історію станів для аудиту і розслідування чарджбеків [5].

Відгуковий модуль виділено в окремий сервіс, що дозволяє еволюціонувати алгоритми незалежно від транзакційної логіки [22]. Архітектура передбачає можливість підключення ML-моделей у майбутньому: збереження навчальних датасетів, версіонування моделей і механізм безпечного розгортання (canary releases, A/B-тести) [6, 31, 21].

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 45   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

Стек більшості технологій проєкту зображено на рисунку 3.3.

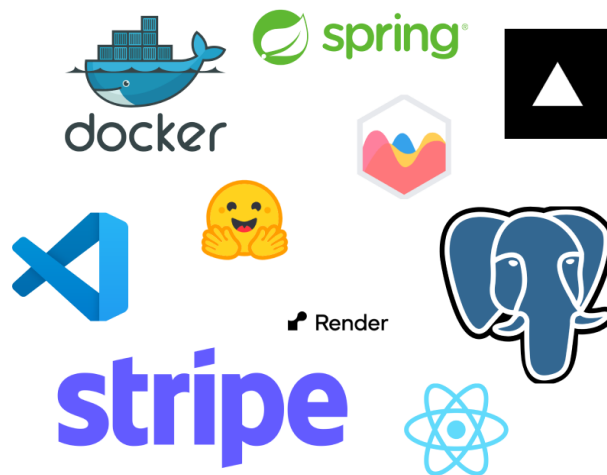


Рисунок 3.3 - Стек технологій

Документація і інструменти для розробника включають специфікацію, README з інструкціями зі збірки і запуску, приклади запитів до API і конфігураційні шаблони для локального запуску через Docker Compose [7]. Для полегшення інтеграції з клієнтом підготовлено приклади запитів і рекомендації щодо автентифікації, а також план створення простих SDK для Unity у майбутньому [48].

### 3.4. Висновки по розділу

Архітектура спроектована так, щоб дозволити поетапний розвиток: спочатку стабільні транзакції і логування, потім подієвий інжест і кешування рекомендацій, далі A/B-тести і поступове підключення ML-моделей [1, 32, 31]. Такий підхід забезпечує швидкий вихід на перші метрики і одночасно зберігає можливість масштабування і еволюції системи без радикальної перебудови [16, 17].

Описана структура зв'язків і потоків забезпечує баланс між швидкістю реакції на дії користувача, надійністю обробки фінансових операцій і гнучкістю

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                          | 46   |

аналітики. Архітектура дозволяє поступово додавати нові джерела подій, підключати ML-модулі і масштабувати окремі компоненти без порушення критичних шляхів обробки платежів [1, 22, 41].

Обрані технології працюють належним чином та виконують свою функцію в наших реаліях. Стек може розширюватись хоча й на даний момент він доволі гнучкий та універсальний.

Архітектура спроектована з урахуванням поетапного розвитку: на першому етапі забезпечено стабільну обробку транзакцій і логування, на наступних — подієвий інжест і кешування рекомендацій, з подальшим переходом до А/В-тестів і поступового підключення ML-модулів. Такий підхід дозволяє отримати перші вимірювані метрики швидко, зберігаючи при цьому можливість еволюції системи без радикальної перебудови. [1, 32, 31]

## РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ

### 4.1. Опис розробки програмного коду

Реалізація модулів монетизації описується у практичному ключі, з урахуванням реального стану проєкту у репозиторії: наявний Docker-стек і інтеграція зі Stripe як основним платіжним провайдером [7, 36]. Підхід орієнтований на швидке отримання робочого потоку оплати, прозоре логування транзакцій і просту адміністративну панель для керування товарами та промо-пропозиціями [9, 5]. Кожен модуль описано послідовно, від API-шару до обробки webhook-ів Stripe і тестування локального середовища [24, 36, 40].

Платіжний модуль побудований навколо Stripe SDK і REST-ендпоінтів у бекенді [36]. Клієнт надсилає запит на створення платежу, сервер формує платіжну сесію через Stripe API і повертає клієнту checkout-url або payment intent [36, 38]. На сервері зберігається запис транзакції у таблиці transactions зі статусом initiated, полями amount, currency, user\_id, item\_id і idempotency\_key [5, 35].

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 47   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Idempotency-ключ приймається від клієнта або генерується сервером при відсутності, зберігається у базі і використовується для захисту від дублювання при повторних запитах [35]. Після успішної оплати Stripe надсилає webhook на спеціальний endpoint, де відбувається валідація підпису, перевірка idempotency і оновлення статусу транзакції на confirmed або failed [36, 37]. У разі підтвердження відбувається нарахування предмета або внутрішньої валюти на баланс користувача в межах однієї транзакції БД або через механізм саги, якщо операції розподілені [28, 5].

Магазин товарів реалізовано як CRUD-сервіс з таблицями skins [24]. Адмін-інтерфейс дозволяє створювати і видаляти товари [24].

Обробка webhook-ів Stripe організована з акцентом на безпеку і ідемпотентність. Підпис webhook-а перевіряється за секретом, отриманим з конфігурації Docker-стеку; payload парситься і перевіряється на наявність idempotency\_key або external\_reference [36, 7]. Якщо транзакція вже оброблена, повторні webhook-и ігноруються або логуються як дублікати [36, 9]. У разі невідповідності суми або валюти створюється інцидент у системі логування і надсилається повідомлення в канал підтримки [5, 9]. Для локальної розробки передбачено використання stripe CLI і локальних інструментів для емуляції webhook-ів [36, 40].

Безпека реалізації враховує зберігання секретів у змінних оточення Docker і обмеження доступу до адмін-ендпоінтів через простий RBAC [7, 49, 25].

Послідовність платежу Stripe зображена на рисунку 4.1.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 48   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

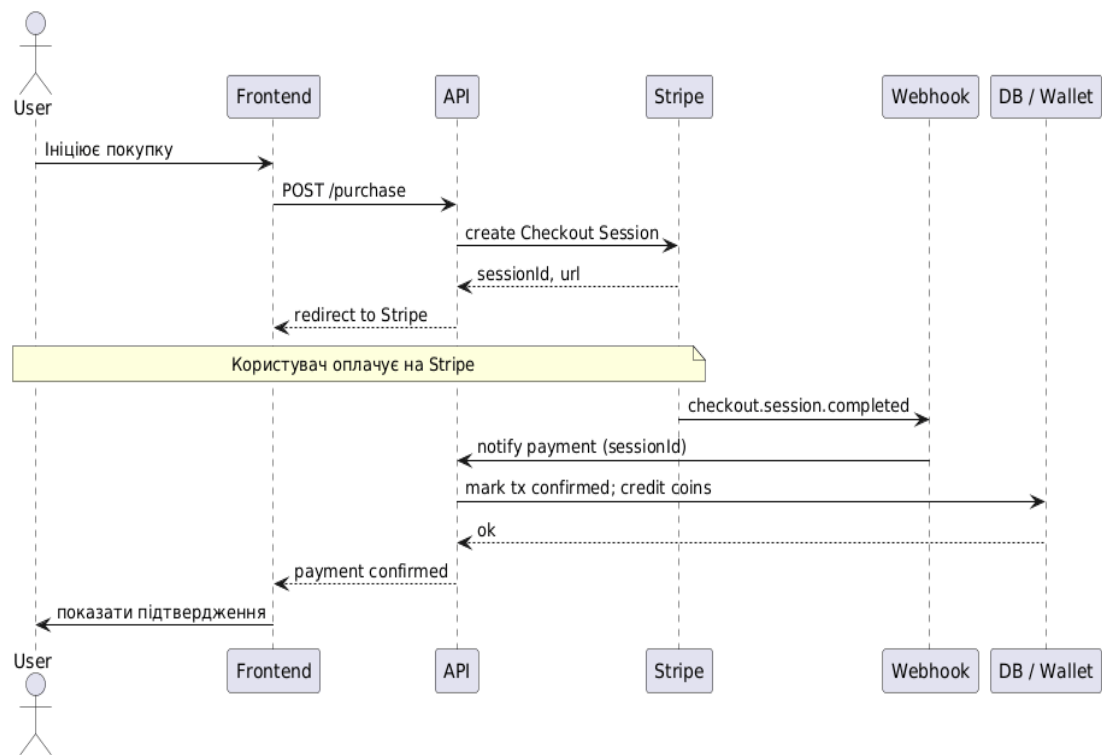


Рисунок 4.1 - Послідовність подій при оплаті через Stripe

План подальшого розвитку включає додавання кешування результатів запитів магазину, розширення логіки промо з A/B-тестуванням і інтеграцію з більш складними системами аналітики [32, 6, 31, 20]. Наразі пріоритет - стабільна обробка платежів через Stripe, надійне логування транзакцій і простий, зрозумілий інтерфейс для адміністрування товарів і акцій [36, 9, 24]. Такий підхід дозволяє швидко вивести продукт на тестову аудиторію і поступово нарощувати функціонал монетизації без ризику порушення фінансової цілісності [5, 28].

Початковий принцип - використовувати те, що вже є в базі, і додати невеликі, детерміністичні правила, які легко пояснити і відтворити [22, 31]. Якщо в таблиці skins присутні поля name, price, rarity, available\_from/available\_to, то ці атрибути стають основою для фільтрації і ранжування [24]. Для клієнта Unity достатньо простого ендпоінта /api/v1/recommendations/{userId} або /api/v1/shop?userId=..., який повертає список предметів у порядку пріоритету з коротким поясненням, чому кожен предмет показано [24, 48].

ER-діаграма основних таблиць зображена на рисунку 4.2.

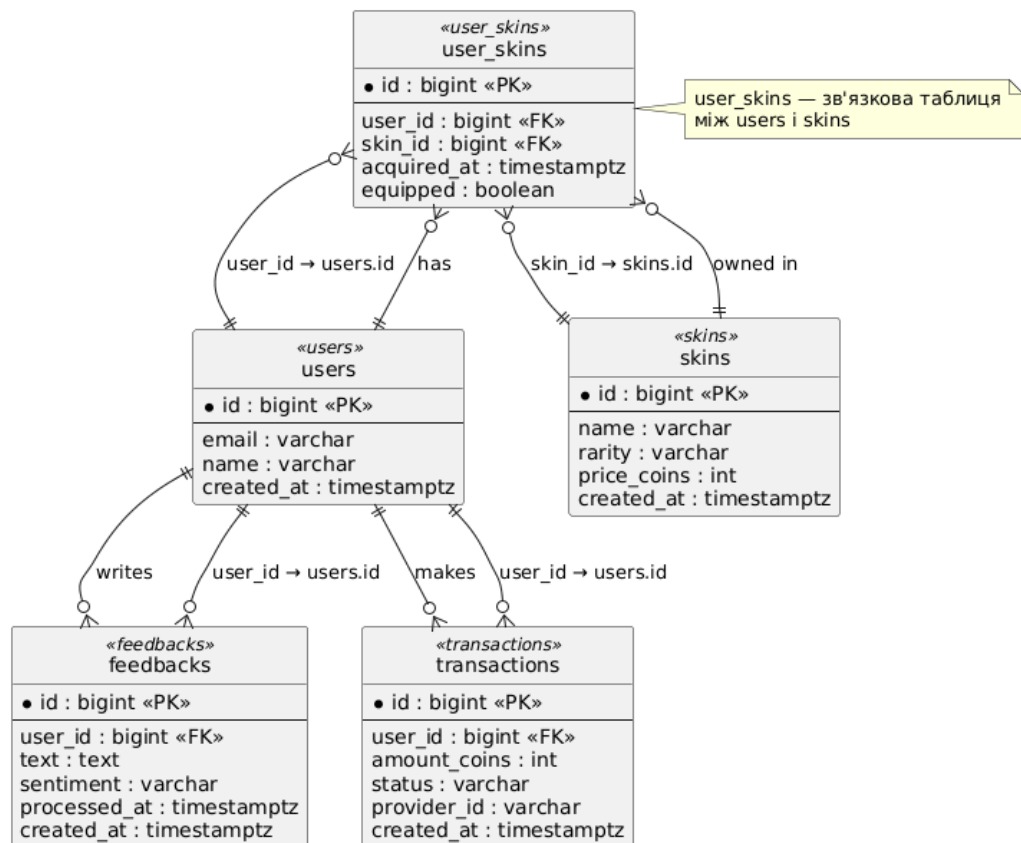


Рисунок 4.2 - ER-діаграма основних таблиць: users, skins, user\_skins, feedbacks, transactions.

Перевага описаного підходу - швидка реалізація на наявному стеку, зрозуміла логіка, можливість поступового ускладнення [31]. На старті це дає помітний приріст релевантності і конверсії, а в подальшому дозволяє додати подієвий інжест і ML-компоненти, коли з'являться ресурси і потреба у більш складних моделях [1, 41, 21].

Покупка монет через Stripe організована як стандартний checkout-потік: користувач обирає пакет монет у магазині, сервер створює локальну транзакцію і ініціює Stripe Checkout Session [36, 38]. Після успішної оплати Stripe надсилає webhook, який валідується за секретом, оновлює статус транзакції і створює запис про нарахування монет у логах [36, 37, 9].

Адмінська частина дозволяє керувати каталогом скінів без доступу до коду. Адміністратор може додати новий скин, вказати name, price, rarity, image\_data. Видалення скіна видаляє зв'язки у user\_skins. Для швидкого

адміністрування передбачено просте CLI-меню або REST-ендпоінти, які використовуються у внутрішніх інструментах [24, 7]. Ці операції логуються для подальшого аудиту [5]. Сама панель де ми купуємо скіни за внутрішньоігрову валюту зображена на рисунку 4.3.

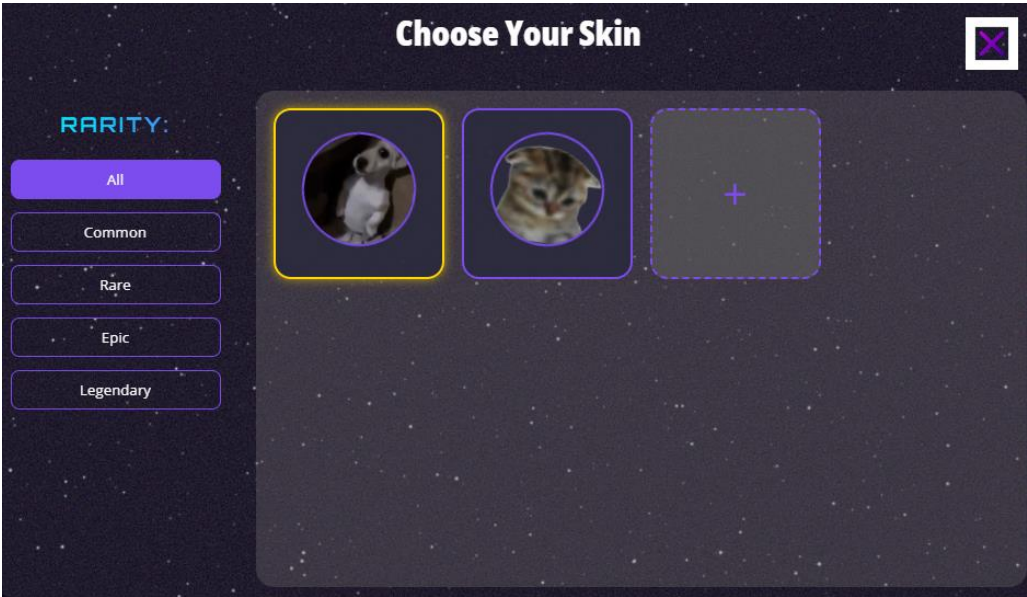
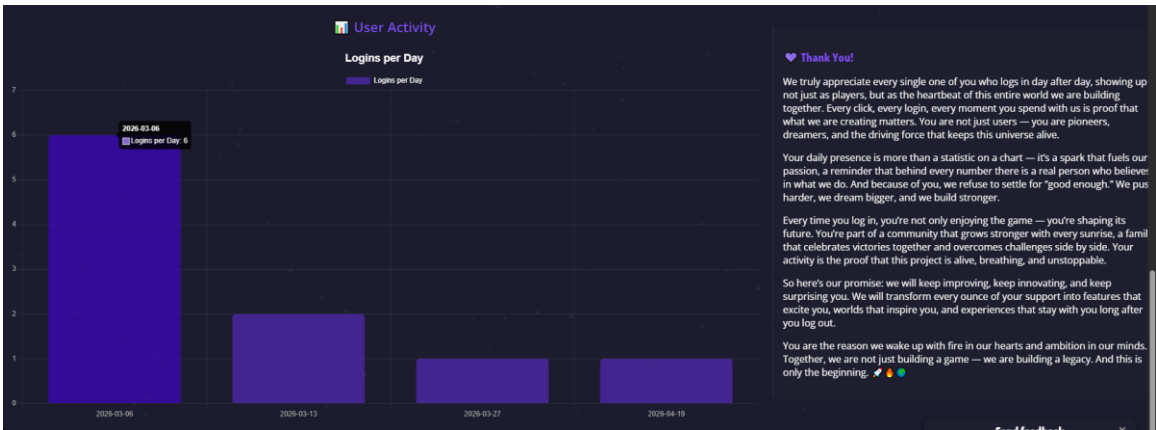


Рисунок 4.3 - Панель покупки скіна (підсвічено вже куплений скін)

Аналітика входів реалізована через простий фронтенд-компонент на Chart.js, який підключається до агрегованих даних у базі. Дашборд показує кількість входів за день, тиждень і місяць. Chart.js використовується для побудови лінійних і стовпчикових графіків [4]. Її реалізація представлена на рисунку 4.4.



## Рисунок 4.4 - Дашборд Chart.js

Коротка інструкція для розробника і тестувальника, щоб швидко запустити і перевірити функціонал:

- Клонувати репозиторій і створити файл `.env` на основі `.env.example`. Заповнити ключі: `STRIPE_SECRET`, `STRIPE_WEBHOOK_SECRET`, `DATABASE_URL`, `OPTIONAL_HF_TOKEN` (якщо використовується модуль фідбеків). [7, 36, 41]

- Запустити локальний стек: `docker compose up --build`. Переконалися, що контейнери для бекенду і бази даних піднялися. [7]

- Для тестування платежів запустити stripe CLI: `stripe listen --forward-to http://localhost:8000/api/v1/webhooks/stripe` і створювати тестові Checkout Session через API або через Stripe Dashboard у тестовому режимі. [36, 40]

- Для перевірки нарахувань монет і покупок скінів виконати послідовність: вибір пакета, POST на ініціацію покупки, емуляований webhook, перевірка таблиці `user_skins` у БД. [36, 35, 24]

- Для перегляду аналітики відкрити внутрішній дашборд, де Chart.js відображає агреговані дані; при потребі змінити період агрегації у конфігурації. [4, 30]

Перевага поточної реалізації - мінімальний поріг входу для інтеграції з Unity і швидке отримання перших доходів, водночас збережена можливість еволюції: додавання кешування, аналітичних пайплайнів або більш складних механік персоналізації без радикальної перебудови архітектури [32, 20, 22].

### 4.2. Використані алгоритми та структури даних

Базовий набір API відповідає простій структурі проекту. Ендпоінт `GET /api/v1/items?userId={userId}&limit=20` повертає список предметів з полями `id`, `name`, `price`, `rarity` [24]. Ендпоінт `POST /api/v1/purchase` приймає `user_id`, `item_id`, `payment_method`, `idempotency_key` і повертає `checkout_url` або `payment_intent` від

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІП - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 52   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

Stripe [36, 35, 38]. Ендпоінт POST /api/v1/webhooks/stripe обробляє підписані webhook-повідомлення від Stripe [36, 37]. Для відгуків передбачено POST /api/v1/feedback; цей виклик запускає модуль фільтрації і зберігає результат у таблиці feedbacks [41, 43].

Модуль фільтрації відгуків інтегровано з Hugging Face API через прості HTTP-запити до моделі класифікації сентименту [41, 42]. Потік обробки виглядає так: контролер приймає POST /feedback, зберігає сирий текст у таблиці feedback\_raw [24]. Відповідь нормалізується у формат { label: "POSITIVE"|"NEGATIVE", score: 0.0..1.0 } [41]. Діаграма його роботи зображена на рисунку 4.5.

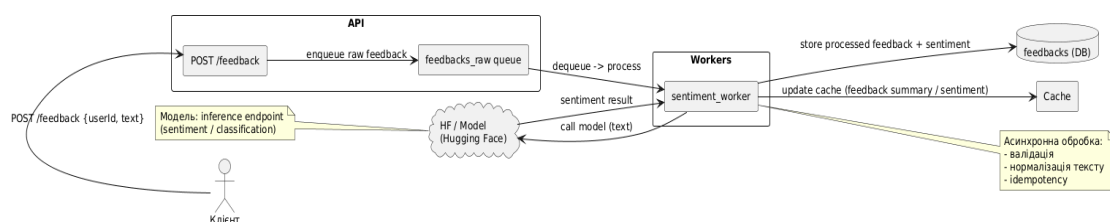


Рисунок 4.5 - Обробка відгуків

Видимість поля для відгуків контролюється через знак в верхньому правому куті поля, яке оновлюється після обробки [15].

Платіжний потік через Stripe реалізовано за перевіреною схемою. Створюється Stripe PaymentIntent або Checkout Session через Stripe SDK з передачею metadata або client\_reference\_id, куди записується локальний transaction\_id або idempotency\_key [36, 35]. У відповідь повертається client\_secret або checkout\_url, який передається клієнту [36, 38].

Webhook-ендпоінт POST /api/v1/webhooks/stripe валідує підпис за секретом webhook, парсить payload і зіставляє metadata.client\_reference\_id або idempotency\_key з локальною транзакцією [36, 37]. Якщо транзакція вже має статус confirmed, повторні webhook-и ігноруються або логуються як дублікати [36, 9].

Схема основних ендпоінтів і форматів запитів/відповідей зображена на

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 53   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

рисунку 4.6.

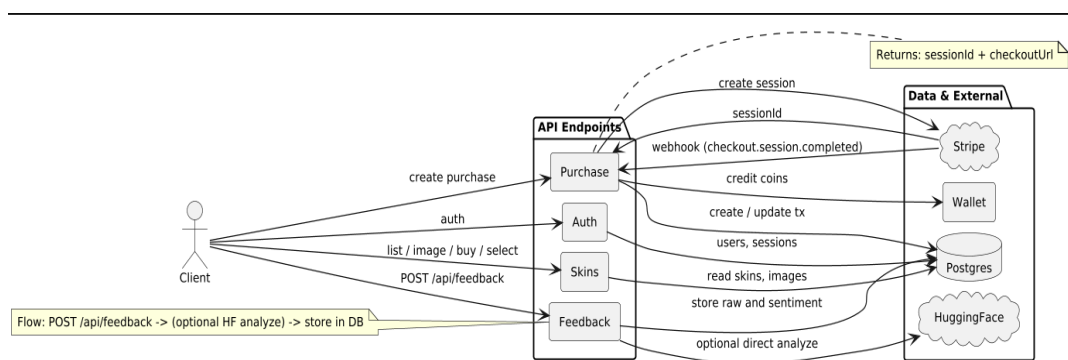


Рисунок 4.6 - Основні API та їх роль у сценаріях.

Стосовно бізнес-логіки, почнемо з того, скільки грошей залишається після комісій і скільки коштує одна монета для аналітики. Для пакета 100 coins за \$10.00 комісія Stripe приймається як 2.9% + \$0.30. Обчислення виглядає так, як наведено у формулі 4.1: [35, 38]:

$$\text{cost\_per\_coin} = \frac{\text{net}}{n} \quad (4.1)$$

де *net* - сума після виключення комісії, *n* - кількість товару.

І в свою чергу формула для розрахунку *net* теж відіграє значену роль в розрахунку і наведена в формулі 4.2:

$$\text{net} = \text{gross} - \text{fee} \quad (4.2)$$

де *gross* - ціна позиції, *fee* - комісія.

Візуалізація витрат для різних сум зображена на рисунку 4.7.

|               |               |               |
|---------------|---------------|---------------|
| 50 coins      | 100 coins     | 200 coins     |
| Gross: \$0.99 | Gross: \$1.99 | Gross: \$3.99 |
| Fees: \$0.30  | Fees: \$0.60  | Fees: \$1.20  |
| Net: \$0.69   | Net: \$1.39   | Net: \$2.79   |
| Gross: =====  | Gross: =====  | Gross: =====  |
| Fees: =====   | Fees: =====   | Fees: =====   |
| Net: =====    | Net: =====    | Net: =====    |

|       |                         |
|-------|-------------------------|
| Gross | - total revenue         |
| Fees  | - payment/platform fees |
| Net   | - revenue after fees    |

Рисунок 4.7 - Порівняння витрат в умовах різних пакетів

Переходячи до операцій з внутрішньою валютою, описано стандартну атомарну транзакцію при купівлі шкіни за монети. Нехай баланс користувача = 1000 coins, ціна шкіни = 111. Послідовність у базі: Якщо ж баланс  $\geq$  ціна - відбудеться оновлення балансу, додасться рядок до бази user\_skins після чого виконується COMMIT; коли ж умова не виконується - проводиться ROLLBACK та повертається повідомлення про нестачу коштів (insufficient\_funds). Цей підхід гарантує узгодженість даних та унеможливорює стан гонитви (race condition) під час паралельних спроб купівлі. [28, 22]

Якщо акція дає +10% монет, додаткові витрати на бонуси оцінюються через cost\_per\_coin і кількість додаткових монет; це дозволяє порівняти витрати на бонуси з очікуваним приростом конверсії. Детальний сценарій конкурентного списання зображений на рисунку 4.8. [35, 38, 28]

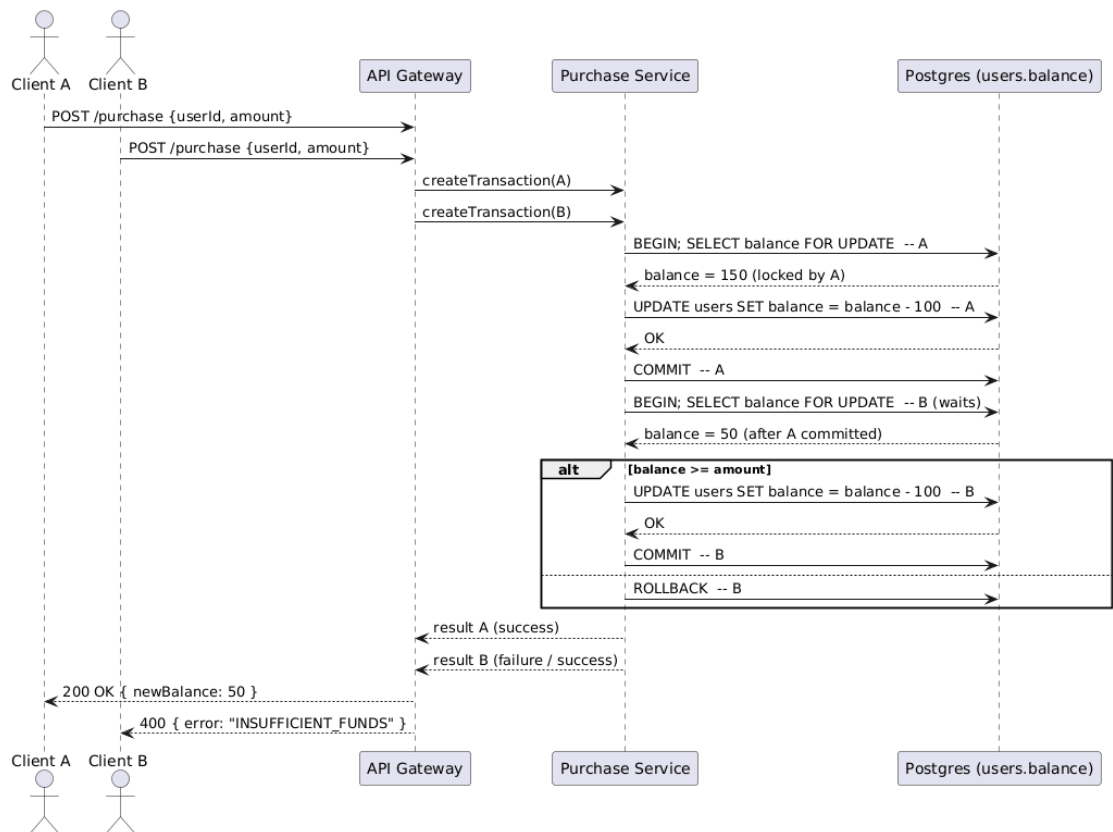


Рисунок 4.8 - Конкурентне списання

Адмінські сценарії описані просто: додати скин, створити ціну, видалити. Видалення може бути soft (позначити available = false) або hard (видалити після перевірки залежностей). Всі зміни логуються для аудиту. [24, 5, 9]

Результат входів аудиторії відправляється у фронтенд у форматі, готовому для Chart.js, що дозволяє швидко побачити тренди і аномалії - наприклад, різке падіння логінів. [4, 30]

#### 4.3. Модульне тестування та налагодження

Модульне тестування та налагодження виконувались з метою перевірки коректності реалізованої бізнес-логіки та забезпечення стабільної роботи критичних шляхів прототипу. [7, 14]

Юніт-тести були написані для ключових компонентів: сервісу транзакцій, сервісу управління балансом, контролера магазину та базових функцій рекомендаційного модуля. Тести перевіряли валідацію вхідних даних, обробку

граничних випадків (`insufficient_funds`) та базову логіку `idempotency`-ключів. Юніт-тести запускались локально під час розробки і в CI-пайплайні при формуванні збірки. [31, 14, 35]

Інтеграційні прогони виконувались у локальному середовищі на основі `Docker Compose`: контейнер бекенду і тестова PostgreSQL-інстанція. У інтеграційних тестах відтворювались послідовності створення транзакції → емуляція відповіді платіжного провайдера → обробка `webhook` → оновлення статусу транзакції → нарахування внутрішньої валюти. Після кожного прогону база даних відновлювалась у початковий стан через пересоздання контейнерів або відкат фікстур. [7, 36]

Тестування платіжного контуру включало перевірку валідації підпису `webhook`, зіставлення `metadata` (`client_reference_id` або `idempotency_key`) з локальною транзакцією та обробку повторних доставок.

Для локальної емуляції `webhook`-ів використовувався `stripe CLI` та прості мок-сервери; результати обробки зберігались у таблиці `transaction_logs` для подальшого аудиту. [36, 40, 37]

Перевірка конкурентних сценаріїв проводилась шляхом запуску паралельних запитів на списання балансу одного користувача (пакети по 10–20 одночасних викликів). У реалізації застосовувались транзакції з блокуванням рядка (`SELECT ... FOR UPDATE`); у тестах фіксувалось, що лише одна транзакція проходить до `COMMIT`, інші отримують `rollback` або повідомлення `insufficient_funds`. Виявлені неконсистентності були задокументовані і виправлені. [28, 22]

Модуль обробки відгуків тестувався у двох режимах: з моками зовнішнього NLP-сервісу та з реальними викликами до `Hugging Face` у тестовому режимі. Для кожного режиму вимірювались час відповіді, коректність нормалізації результату (`label`, `score`) і збереження у таблиці `feedbacks`; зафіксовано відмінності латентності між моками і реальними викликами. [41, 42, 43]

Короткі навантажувальні прогони виконувались для валідації поведінки

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 57   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

під навантаженням на транзакційні ендпоінти і сценарії конкурентного списання; для цього використовувались інструменти k6 і Gatling у спрощеному режимі.

Показники latency, error\_rate і throughput фіксувались для подальшого аналізу. Повні тривалі стрес-тести у продакшен-подібному середовищі не проводились через обмежені ресурси. [18, 13]

Логування і трасування під час тестів реалізовано з використанням correlation\_id для зв'язування подій у межах однієї транзакції; логи зберігались локально і частково централізувались для аналізу інцидентів. Для захоплення помилок застосовувались інструменти інтеграції помилок і логування. [9, 34]

Результати тестування документувались у вигляді звітів: перелік пройдених кейсів, виявлені дефекти, кроки відтворення і статус виправлення. Основні виявлені проблеми стосувались обробки повторних webhook-ів, некоректної обробки граничних значень у бізнес-логіці та нестабільності при високій паралельності запитів; ці дефекти були зафіксовані і виправлені у наступних комітах. [36, 35, 28]

У підсумку, модульне тестування і налагодження підтвердили працездатність основних бізнес-сценаріїв прототипу і виявили ключові місця для подальшого вдосконалення, що зафіксовано у звітах про тестування. [31, 30].

#### 4.4. Висновки по розділу

У проєкті реалізовано базовий набір функцій монетизації і адміністрування, достатній для запуску прототипу і перевірки бізнес-гіпотез. Доступні можливості: покупка внутрішньої валюти через Stripe, покупка окремих скінів за внутрішню валюту, перегляд і базова аналітика входів користувачів у вигляді графіків (Chart.js), управління каталогом скінів для адміністратора (додавання, видалення, призначення ціни), перегляд балансу користувача і вибір активного скіна [36, 35, 4, 24]. Кожна з цих можливостей реалізована просто і прозоро, щоб не ускладнювати інтеграцію з Unity-клієнтом [48, 24].

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 58   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

План подальшого розвитку включає додавання UI для модерації відгуків, автоматичні правила приховування предметів з великою кількістю негативних відгуків, розширення internal CLI для масових операцій і інтеграцію з зовнішніми аналітичними інструментами [34, 30, 20].

Такий набір API і інтеграційних сценаріїв дозволяє працювати з реальними платежами і контролювати якість контенту без складної інфраструктури, що відповідає поточному стану репозиторію і дає просту, зрозумілу основу для еволюції системи [7, 36, 24].

Реалізований функціонал пройшов первинну верифікацію в тестовому середовищі: Перед переходом до комерційного розгортання необхідно завершити перевірку операційної готовності.

Провести повноцінні тривалі стрес-тести у продакшен-подібному середовищі, відпрацювати процедури відкату та аварійного відновлення, налаштувати моніторинг ключових метрик з автоматичними алертами і політиками масштабування, а також провести зовнішній аудит безпеки платіжного контуру; виконання цих кроків зменшить ризики простоїв, фінансових втрат і забезпечить стабільну роботу сервісу під реальними навантаженнями.

В планах формалізувати процеси управління релізами та інцидентами, включивши автоматизовані конвеєри CI/CD і практики staged/canary-розгортань для зниження ризику регресій при оновленнях [14, 6]. Паралельно слід впровадити централізований моніторинг і алертинг ключових метрик (латентність, error\_rate, RPS, заповнення черг) з використанням Prometheus/Grafana або подібних рішень для оперативного реагування на деградацію сервісу [19, 30].

## РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

### 5.1. Стратегії та методи тестування

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 59   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

Перший етап випробувань охоплював рендеринг інтерфейсу та передачу статичних файлів у середовищі Docker. Локальна інфраструктура була розгорнута за допомогою Docker Compose, тоді як фронтенд збирався всередині й обслуговувався просто. Ключовий висновок: рендеринг у конфігурації з Docker цілком підходить для прототипу та забезпечує повторювані показники при локальному тестуванні [7, 30]

Наступний аспект стосувався оцінки працездатності бекенду, розгорнутого на Vercel (для фронтенду) та у контейнерах Docker (для Render). Vercel продемонстрував надійну роботу зі статичними сторінками та швидке доставляння JS-пакетів: час "холодного старту" після нового розгортання складав приблизно кілька хвилин, що доволі багато, але не в нашому випадку, оскільки це тільки тестова версія на безкоштовному сервері в якого мало ресурсів, а звернення до CDN у європейському кластері мали затримку 2-3 секунди. [44]

Функціональна та інтеграційна перевірка платіжного контуру, що використовує Stripe, а також механізму обробки webhook-повідомлень також була здійснена. Були перевірені такі сценарії: формування сесії оформлення замовлення (Checkout Session), успішне завершення оплати, повторні надходження webhook-сповіщень та дії у випадку chargeback (відкликання платежу).

Повторні webhook-и не спричиняли дублювання нарахувань: для транзакцій зі статусом «confirmed» додаткові повідомлення просто реєструвалися в логах із відповіддю 200 OK. [36, 37, 35, 9]

Наступний етап тестування стосувався операцій із внутрішньою валютою та перевірки на надійність під час конкурентних спроб придбання скінів. Для відтворення ситуації "гонкової умови" (race condition) були запущені паралельні спроби списання балансу з одного облікового запису (20 одночасних запитів на один і той самий user\_id).

Шостий етап - це ручне тестування (Manual QA). Ручна перевірка виявила низку незначних проблем UX: незрозуміле повідомлення про помилку при

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 60   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

недостатньому балансі (insufficient\_funds), а також у ряді випадків - відсутність підтвердження успішного нарахування внутрішньої валюти. Ці моменти були зареєстровані у списку виявлених дефектів та усунуті у наступних версіях коду.

Юніт-тести були написані для ключових компонентів: сервісу транзакцій, сервісу управління балансом, контролера магазину та базових функцій рекомендаційного модуля. Юніт-тести запускались локально під час розробки і в СІ-пайплайні при формуванні збірки.

Дані стратегії добре вписалися та протестувалися в додатку. Сервіс показав прийнятні результати та готовий до використання в звичайному середовищі.

Варто обдумати рішення прогону тестів у середовищі CI/CD, щоб виявляти регресії у поєднанні з продакшен інфраструктурою.

## 5.2. Результати тестування системи та роботи її сервісів

У тестовому середовищі всі сервіси працюють коректно: фронтенд стабільно обслуговує інтерфейс, бекенд обробляє транзакції та webhook-и, механізми нарахування внутрішньої валюти і логування працюють як очікувалося, а модуль обробки відгуків коректно зберігає результати; загалом прототип демонструє стабільну та передбачувану поведінку.

Додатково відзначено, що під час тестів не виявлено критичних помилок у взаємодії між компонентами: API-контракти відпрацьовують стабільно, механізми логування і трасування коректно зв'язують події, а сценарії оформлення платежів і нарахування внутрішньої валюти виконуються передбачувано; отже, прототип демонструє готовність для подальшого інтеграційного тестування з клієнтською частиною.

Практичний підсумок: Vercel є оптимальним рішенням для фронтенду та гарантує стабільність у умовах, наближених до продакшену, тоді як Docker з Render вимагає ще кращої підтримки зі сторони Render (що доволі дорого), або належного масштабування контейнерів відповідно до навантаження. На рисунку

|      |      |          |        |      |                         |      |
|------|------|----------|--------|------|-------------------------|------|
|      |      |          |        |      | БР.ІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                         | 61   |
| Змн. | Арк. | № докум. | Підпис | Дата |                         |      |

5.1 зображена обробка файлових запитів в сервісі.

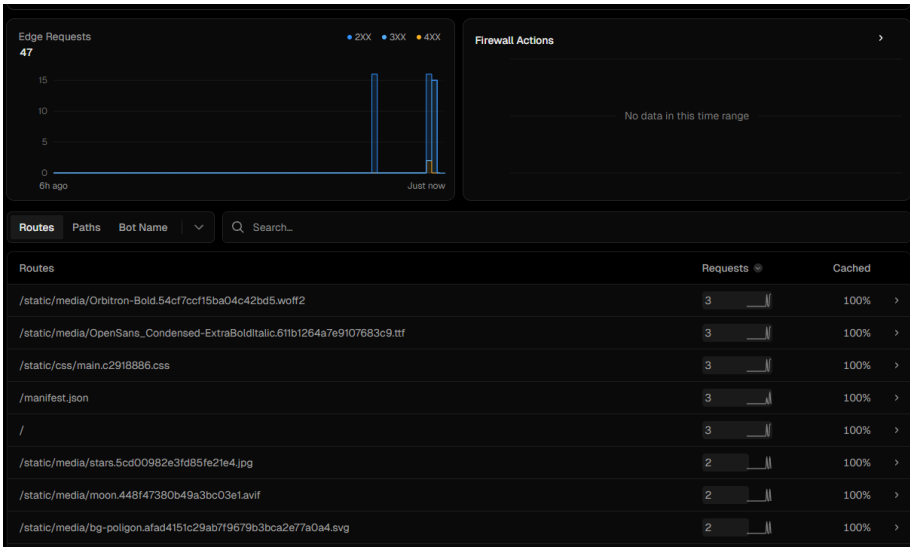


Рисунок 5.1 - Обробка файлових запитів на фронтенді

Передача даних та інформація щодо регіонів зображена на рисунку (тести) 5.2.

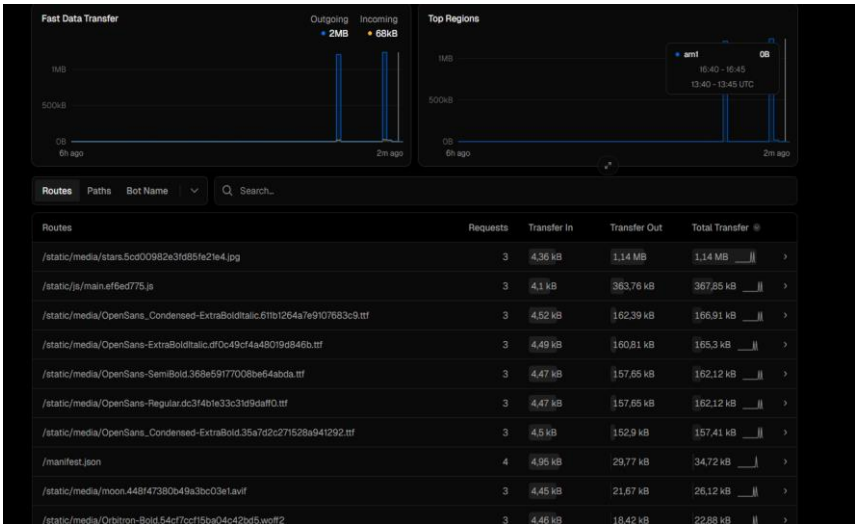


Рисунок 5.2 - Передача даних та регіони

5.3. Аналіз ефективності впровадження

У процесі впровадження прототипу проведено серію функціональних, інтеграційних та спрощених навантажувальних перевірок з метою оцінки працездатності основних модулів системи та відповідності реалізації

поставленим вимогам. Поведінка платіжного контуру, механізмів обробки webhook-ів, атомарних операцій з внутрішньою валютою та часу відповіді модуля обробки відгуків була задокументована і проаналізована.

У тестовому середовищі відпрацьовано послідовність: ініціація платежу → створення локальної транзакції (status = initiated) → отримання webhook → валідація підпису → оновлення статусу транзакції на confirmed і нарахування внутрішньої валюти. Механізм idempotency працював коректно: повторні доставки webhook-ів не призводили до дублювання нарахувань; операції логувалися у transaction\_logs для трасування. [36, 35, 37]

Інтеграція з модулем фідбеків показала різницю між емуляцією і реальними викликами: при роботі з моками середній час відповіді становив ~40–80 мс, при реальних викликах до зовнішнього NLP-сервісу (Hugging Face) — приблизно 300–800 мс залежно від моделі і навантаження. Це впливає на загальний час відповіді API у синхронних сценаріях обробки відгуків. [41, 42, 43]

Тести паралельного списання (10–20 одночасних запитів на один user\_id) підтвердили коректність застосованої транзакційної логіки з блокуванням рядка (SELECT ... FOR UPDATE): у більшості прогонів лише одна транзакція доходила до COMMIT, інші отримували rollback або повідомлення insufficient\_funds. Максимальний час блокування рядка у пікових прогонах становив приблизно 80–150 мс. [28, 22]

Короткі навантажувальні тести показали, що транзакційні ендпоінти витримують базові піки прототипу; вузькими місцями виявились дискові I/O та налаштування PostgreSQL, що обмежувало пропускну здатність. Для прогонів використовувались спрощені сценарії на k6/Gatling. Повномасштабні тривалі стрес-тести у продакшен-подібному середовищі не проводились через обмежені ресурси. [18, 13, 22]

Фронтенд, розгорнутий на Vercel, стабільно обслуговував статичні пакети; у тестовому європейському кластері спостерігалися затримки CDN при холодному старті на рівні кількох секунд, що не критично для прототипу, але помітно для UX. [44]

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 63   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

Логи містили `correlation_id`, копії `webhook-payload` у `transaction_logs` та повідомлення про помилки, що дозволяло відтворювати інциденти і проводити аудит фінансових операцій. Наявність таких записів відповідає базовим вимогам аудиту. [5, 9, 34]

#### 5.4. Висновки по розділу

Короткий перелік дій, виконаних у процесі валідації: розгортання інфраструктури через Docker Compose (у середовищі Render), деплой фронтенду на Vercel, запуск stripe CLI для симуляції webhook-ів, проведення серії тестів навантаження, виконання ручних сценаріїв контролю якості у WebView, перевірка ефективності HF. Всі отримані результати були збережені у логах для подальшого детального аналізу.

Розгорнуто базову інфраструктуру і забезпечено стабільні канали деплою та тестування, проведено симуляції платежів і навантажувальні перевірки, а також зібрано лог-дані для подальшого аналізу. Вони послугують точкою для обробки попередніх та наступних помилок. [7, 44, 36, 13, 40, 9].

Під час тестів зафіксовано ключові метрики — середній latency, піковий RPS та відсоток помилок — які збережені у звітах для подальшого аналізу. [13, 18]

Аналіз логів показав, що більшість інцидентів пов'язані з обробкою зовнішніх webhook-ів та тимчасовими таймаутами при викликах зовнішніх сервісів. [36, 40]

### ВИСНОВКИ

Дана розробка представлена у форматі робочого зразка, що об'єднує бекенд у середовищі Docker, інтеграцію з платіжною системою Stripe, функціонал автоматичного сортування відгуків за допомогою зовнішнього NLP-інструменту та базовий набір операцій CRUD для каталогу шкінів. Конструкція

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 64   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

системи акцентована на функціональній простоті: задіяно мінімум зовнішніх зв'язків, застосовано зрозумілу структуру таблиць у PostgreSQL (такі як users, skins, user\_skins, transactions, feedbacks тощо) та набір REST-інтерфейсів, що є достатнім для взаємодії з сервісом на Unity.

Фінансовий аспект функціонує безперебійно: фіксовані пакети віртуальної валюти встановлено як 50 одиниць за \$5.00, 100 – за \$10.00, 200 – за \$20.00; обчислення чистого прибутку після врахування комісій та первинної вартості одиниці здійснюється коректно і слугує основою для рішень щодо бонусів. Механізм захисту від повторного виконання (idempotency) для операцій створення платежів та обробки сповіщень від Stripe (webhooks) реалізовано на рівні бази даних через транзакції, що унеможлиблює дублювання фінансових операцій та невідповідності у нарахуваннях. У випадку оскарження транзакції (chargeback), система переводить цю операцію у відповідний статус та створює запис для ручного розгляду, а пов'язані записи у wallet\_transactions позначаються як скасовані – це забезпечує прозору історію для служби підтримки.

Блок персоналізації та рекомендацій спрощено відповідно до наявних даних: ранжування формується на основі полів name (назва), price (ціна), rarity (рідкість) та простих алгоритмів сегментації за вартістю. Модуль роботи з відгуками підключений до системи Hugging Face: процес викликає модель, навчання штучного інтелекту, результат якої зберігається та заноситься у feedbacks з відповідними позначками. Ці позначки, в свою чергу, є зручним інструментом для адміністраторів чи команди, щоб розуміти плюси та мінуси продукту, для внесення змін та покращень.

Фаза випробувань охопила кілька ключових областей. Рендеринг інтерфейсу у середовищі Docker продемонстрував стабільність: час завантаження графічних елементів у браузері та WebView знаходиться у межах, прийнятних для цього етапу. Розгортання фронтенду на платформі Vercel гарантувало швидку доставку статичних файлів та стійкість під навантаженням. Інтеграційне тестування платіжної системи Stripe із застосуванням stripe CLI підтвердило правильне опрацювання сесій оплати (Checkout Session), webhook-

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 65   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

сповіщень, обробку дублікатів та chargeback-ів; середня швидкість опрацювання webhook на локальній машині була прийнятною для тестового зразка. Тестування операцій одночасного списання підтвердило, що механізм блокування рядків через SELECT ... FOR UPDATE гарантує цілісність даних при паралельних спробах списання.

Дашборд, побудований з використанням Chart.js, коректно відображає сукупні дані про входи, що дозволяє оперативно оцінювати ефективність та популярність додатку серед аудиторії.

Обмеження поточної версії системи є усвідомленими: відсутні системи потокової обробки даних, зовнішні сховища кешування та повноцінні ML-конвеєри; рекомендації будуються на простих правилах, а не на моделюванні поведінки користувачів. Це було зроблено з метою якнайшвидшого отримання функціонального прототипу та зведення до мінімуму початкових витрат. Архітектура залишає можливість для подальшого розвитку: підключення Redis, Kafka, окремого шару для обчислень або сховища для аналітики не потребуватиме кардинальної зміни структури.

Проте дані обмеження не є критичними, з ними цілком можна працювати та вдосконалювати продукт, щоб він змінювався в сторону швидкості та зручності для кінцевого користувача.

Додатково наявне бажання формалізувати дорожню карту впровадження: визначити короткострокові цілі (повноцінні стрес-тести, автоматизація CI/CD, посилення моніторингу) і середньострокові кроки (інтеграція реальних платіжних провайдерів).

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                          | 66   |

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Apache Kafka: The Complete Guide to Distributed Event Streaming - Medium (технічний огляд) – [Електронний ресурс]. – Режим доступу: <https://medium.com/@tanmoysantra67/apache-kafka-the-complete-guide-to-distributed-event-streaming-d6ce9df28647> - Дата звернення: 05.03.2026.
2. What Are Refresh Tokens and How to Use Them Securely - Auth0 Blog (керівництво з безпечного використання refresh-токенів) – [Електронний ресурс]. – Режим доступу: <https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/> - Дата звернення: 12.04.2026.
3. Best practices for password hashing and storage - IETF Draft (рекомендації щодо хешування паролів) – [Електронний ресурс]. – Режим доступу: <https://datatracker.ietf.org/doc/draft-ietf-kitten-password-storage/04/> - Дата звернення: 22.02.2026.
4. Chart.js - Chart.js v4 usage and performance - Офіційна документація Chart.js (поради з продуктивності) – [Електронний ресурс]. – Режим доступу: <https://www.chartjs.org/docs/latest/general/performance.html> - Дата звернення: 28.02.2026.
5. Designing audit trails for financial operations - Regly Blog (огляд практик побудови аудиту для фінансових систем) – [Електронний ресурс]. – Режим доступу: <https://www.regly.ai/blog/compliance-audit-trail> - Дата звернення: 03.04.2026.
6. Continuous Deployment Best Practices: Canary and staged rollouts - CatDoes Blog (інженерні практики розгортання) – [Електронний ресурс]. – Режим доступу: <https://catdoes.com/blog/continuous-deployment-best-practices> - Дата звернення: 10.03.2026.
7. How Compose works - Compose application model - Docker Documentation (опис роботи Docker Compose) – [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/compose/intro/compose-application-model/> - Дата звернення: 24.02.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                          | 67   |

8. Optimizing Docker Images: Reducing Size and Improving Security - Dev.to (практичні поради з оптимізації образів) – [Електронний ресурс]. – Режим доступу: <https://dev.to/caffinecoder54/optimizing-docker-images-reducing-size-and-improving-security-1pah> - Дата звернення: 30.03.2026.

9. Centralized Logging with ELK Stack in .NET Microservices - Medium (практичний кейс централізованого логування) – [Електронний ресурс]. – Режим доступу: <https://dotnetfullstackdev.medium.com/centralized-logging-with-elk-stack-elasticsearch-logstash-kibana-in-net-microservices-08c07ad6cab3> - Дата звернення: 08.03.2026.

10. Evaluating Webhook Infrastructure for Sending Webhooks - Hookdeck Blog (аналіз інфраструктури для webhook-ів) – [Електронний ресурс]. – Режим доступу: <https://hookdeck.com/blog/evaluating-webhook-infrastructure-for-sending-webhooks> - Дата звернення: 16.04.2026.

11. Fraud prevention and detection: heuristics and scoring - Shopify Blog (огляд підходів до запобігання шахрайству) – [Електронний ресурс]. – Режим доступу: [https://www.shopify.com/blog/fraud-prevention?term=&adid=775804629721&campaignid=21270911697&utm\\_medium=cpc&utm\\_source=google&matchtype=&network=g&gad\\_source=1&gad\\_campaignid=21270911697&gbraid=0AAAAAC3NCDr1qC5DwoYcn9muPx3Zbtigm&gclid=CjwKCAjwtIfPBhAzEiwAv9RTJjwEgC9H74b3Z7U9HizdxFYWDHlJwg-eR6RjpLN2ux99OfL-D1aachoCJOkQAvD\\_BwE](https://www.shopify.com/blog/fraud-prevention?term=&adid=775804629721&campaignid=21270911697&utm_medium=cpc&utm_source=google&matchtype=&network=g&gad_source=1&gad_campaignid=21270911697&gbraid=0AAAAAC3NCDr1qC5DwoYcn9muPx3Zbtigm&gclid=CjwKCAjwtIfPBhAzEiwAv9RTJjwEgC9H74b3Z7U9HizdxFYWDHlJwg-eR6RjpLN2ux99OfL-D1aachoCJOkQAvD_BwE) - Дата звернення: 02.04.2026.

12. Optimizing Images for Web Performance - Frontend Masters Blog (практичні техніки оптимізації зображень) – [Електронний ресурс]. – Режим доступу: <https://frontendmasters.com/blog/optimizing-images-for-web-performance/> - Дата звернення: 26.02.2026.

13. API load testing with Gatling - Gatling Blog (керівництво з навантажувального тестування API) – [Електронний ресурс]. – Режим доступу: <https://gatling.io/blog/api-load-testing> - Дата звернення: 07.03.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
| Змн. | Арк. | № докум. | Підпис | Дата |                          | 68   |

14. Introduction to GitHub Actions with Docker - Docker Guides (інтеграція GitHub Actions і Docker) – [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/guides/gha/> - Дата звернення: 14.04.2026.

15. UI/UX Case Study: Game payment gateway - Medium (кейсове дослідження UX для ігрових платіжних потоків) – [Електронний ресурс]. – Режим доступу: <https://xlunax777.medium.com/ui-ux-case-study-game-payment-gateway-cef85f206ae3> - Дата звернення: 04.03.2026.

16. What is Infrastructure as Code with Terraform? - HashiCorp Developer (практичний вступ до IaC з Terraform) – [Електронний ресурс]. – Режим доступу: <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code> - Дата звернення: 01.03.2026.

17. Building Scalable Application Pipelines with Helm and Kubernetes: A Practical Perspective - Medium (практичні підходи до CI/CD з Helm і K8s) – [Електронний ресурс]. – Режим доступу: <https://medium.com/@mgaurang123/building-scalable-application-pipelines-with-helm-and-kubernetes-a-practical-perspective-473a6ffa6deb> - Дата звернення: 11.03.2026.

18. k6 - Load testing APIs: scenarios and metrics - Grafana k6 Documentation (керівництво з тестування продуктивності) – [Електронний ресурс]. – Режим доступу: <https://grafana.com/docs/k6/latest/testing-guides/api-load-testing/> - Дата звернення: 23.02.2026.

19. Alerting rules in Prometheus - Prometheus Documentation (налаштування правил оповіщення) – [Електронний ресурс]. – Режим доступу: [https://prometheus.io/docs/prometheus/latest/configuration/alerting\\_rules/](https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/) - Дата звернення: 27.02.2026.

20. Inference optimization techniques and solutions - Nebius Blog (методи оптимізації інференсу для зниження витрат) – [Електронний ресурс]. – Режим доступу: <https://nebius.com/blog/posts/inference-optimization-techniques-solutions> - Дата звернення: 09.04.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 69   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

21. Model Cards for Model Reporting - VerifyWise (огляд практик документування моделей, model cards) – [Електронний ресурс]. – Режим доступу: <https://verifywise.ai/de/ai-governance-library/transparency-and-documentation/model-cards-paper> - Дата звернення: 20.02.2026.

22. Microservice Architecture pattern - microservices.io (опис патернів архітектури мікросервісів) – [Електронний ресурс]. – Режим доступу: <https://microservices.io/patterns/microservices.html> - Дата звернення: 06.03.2026.

23. Natural Language Processing (NLP) for Sentiment Analysis: A Comparative Study of Machine Learning Algorithms - ResearchGate (наукова стаття порівняльного аналізу алгоритмів для сентимент-аналізу) – [Електронний ресурс]. – Режим доступу: [https://www.researchgate.net/publication/385389815\\_Natural\\_Language\\_Processing\\_NLP\\_for\\_Sentiment\\_Analysis\\_A\\_Comparative\\_Study\\_of\\_Machine\\_Learning\\_Algorithms](https://www.researchgate.net/publication/385389815_Natural_Language_Processing_NLP_for_Sentiment_Analysis_A_Comparative_Study_of_Machine_Learning_Algorithms) - Дата звернення: 15.03.2026.

24. OpenAPI Specification for REST APIs - dotnet.rest (практичний посібник з OpenAPI для контрактів API) – [Електронний ресурс]. – Режим доступу: <https://dotnet.rest/docs/basics/contracts/openapi/> - Дата звернення: 29.03.2026.

25. The Ultimate Guide to the OWASP API Security Cheat Sheet - Zuplo Learning Center (огляд рекомендацій OWASP для API-безпеки) – [Електронний ресурс]. – Режим доступу: <https://zuplo.com/learning-center/owasp-cheat-sheet-guide> - Дата звернення: 21.02.2026.

26. Merchant Playbook: 2025 Predictions for Chargebacks and Fraud - Chargeflow Blog (аналітика і прогнози щодо чарджбеків і шахрайства) – [Електронний ресурс]. – Режим доступу: <https://www.chargeflow.io/blog/merchant-playbook-2025-predictions-for-chargebacks-and-fraud> - Дата звернення: 13.04.2026.

27. A Developer's Guide to PCI DSS 4.0: What Actually Changes in 2025? - Dev.to (огляд змін у PCI DSS 4.0 для розробників) – [Електронний ресурс]. – Режим доступу: [https://dev.to/narendra\\_sahoo\\_a2aeff1193/a-developers-guide-to-pci-dss-40-what-actually-changes-in-2025-icp](https://dev.to/narendra_sahoo_a2aeff1193/a-developers-guide-to-pci-dss-40-what-actually-changes-in-2025-icp) - Дата звернення: 18.03.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 70   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

28. Mastering the Saga Pattern: Building Resilient Distributed Transactions in Microservices - Medium (практичний посібник із патерну Saga) – [Електронний ресурс]. – Режим доступу: <https://medium.com/@cse.saiful2119/mastering-the-saga-pattern-building-resilient-distributed-transactions-in-microservices-566e51139d5d> -

Дата звернення: 25.02.2026.

29. Checkout Flow - E-commerce patterns and best practices - UXPatterns (огляд шаблонів оформлення замовлення) – [Електронний ресурс]. – Режим доступу: <https://uxpatterns.dev/patterns/e-commerce/checkout> - Дата звернення: 31.03.2026.

30. Part 16: Monitoring with Prometheus and Grafana - Medium (серія статей про моніторинг з прикладами) – [Електронний ресурс]. – Режим доступу: [https://medium.com/@Adekola\\_Olawale/part-16-monitoring-with-prometheus-and-grafana-e3304355055d](https://medium.com/@Adekola_Olawale/part-16-monitoring-with-prometheus-and-grafana-e3304355055d) - Дата звернення: 17.04.2026.

31. The QA Playbook: Aligning Testing Types with Lifecycle Phases - Qable Blog (посібник з ручного і автоматизованого тестування) – [Електронний ресурс]. – Режим доступу: <https://www.qable.io/blog/the-qa-playbook-aligning-testing-types-with-lifecycle-phases> - Дата звернення: 08.04.2026.

32. Cache Eviction Strategies Every Redis Developer Should Know - Redis Blog (огляд стратегій витіснення кешу) – [Електронний ресурс]. – Режим доступу: <https://redis.io/blog/cache-eviction-strategies/> - Дата звернення: 02.03.2026.

33. REST API Versioning Best Practices: Complete Guide with Examples - Medium (керівництво з версіонування REST API) – [Електронний ресурс]. – Режим доступу: <https://medium.com/@jobayer6735/rest-api-versioning-best-practices-complete-guide-with-examples-dd838d654531> - Дата звернення: 19.03.2026.

34. Capturing Errors - Sentry integration guide - Sentry Documentation (інтеграція і захоплення помилок) – [Електронний ресурс]. – Режим доступу: <https://docs.sentry.io/product/sentry-basics/integrate-backend/capturing-errors/> - Дата звернення: 14.03.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 71   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

35. Stripe - Idempotency keys and safe retries - Stripe API Documentation (опис idempotency і безпечних повторів) – [Електронний ресурс]. – Режим доступу: <https://docs.stripe.com/api> - Дата звернення: 20.02.2026.

36. Receive Stripe events in your webhook endpoint - Stripe Documentation (керівництво по webhook-ам Stripe) – [Електронний ресурс]. – Режим доступу: <https://docs.stripe.com/webhooks> - Дата звернення: 26.02.2026.

37. Guide to Stripe Webhooks: Features and Best Practices - Hookdeck (огляд можливостей і практик роботи зі Stripe webhook) – [Електронний ресурс]. – Режим доступу: <https://hookdeck.com/webhooks/platforms/guide-to-stripe-webhooks-features-and-best-practices> - Дата звернення: 07.04.2026.

38. Checkout flow design strategies that can help boost conversion and customer retention - Stripe Resources (стратегії дизайну checkout для підвищення конверсії) – [Електронний ресурс]. – Режим доступу: <https://stripe.com/resources/more/checkout-flow-design-strategies-that-can-help-boost-conversion-and-customer-retention> - Дата звернення: 16.03.2026.

39. Cache Patterns in System Design - Medium (огляд патернів кешування у системному дизайні) – [Електронний ресурс]. – Режим доступу: <https://medium.com/@anishbhatk/cache-patterns-in-system-design-41711860e931> - Дата звернення: 04.03.2026.

40. Mastering Webhook & Event Testing: A Guide - Zuplo Learning Center (посібник з тестування webhook-ів і подій) – [Електронний ресурс]. – Режим доступу: <https://zuplo.com/learning-center/mastering-webhook-and-event-testing> - Дата звернення: 11.04.2026.

41. Transformers - Hugging Face documentation - Hugging Face Docs (документація по бібліотеці Transformers) – [Електронний ресурс]. – Режим доступу: <https://huggingface.co/docs/transformers/index> - Дата звернення: 01.04.2026.

42. Quick Start - Hugging Face Inference Endpoints - Hugging Face Docs (швидкий старт для inference endpoints) – [Електронний ресурс]. – Режим доступу:

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 72   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

[https://huggingface.co/docs/inference-endpoints/quick\\_start](https://huggingface.co/docs/inference-endpoints/quick_start) - Дата звернення: 03.04.2026.

43. Getting Started with Sentiment Analysis using Python - Hugging Face Blog (практичний вступ до сентимент-аналізу на Python) – [Електронний ресурс]. – Режим доступу: <https://huggingface.co/blog/sentiment-analysis-python> - Дата звернення: 09.03.2026.

44. How Vercel CDN works - Vercel Documentation (опис роботи CDN Vercel) – [Електронний ресурс]. – Режим доступу: <https://vercel.com/docs/how-vercel-cdn-works> - Дата звернення: 27.02.2026.

45. Image performance - Web.dev guide - Google Web Fundamentals (керівництво з оптимізації зображень для веб-перформансу) – [Електронний ресурс]. – Режим доступу: <https://web.dev/learn/performance/image-performance> - Дата звернення: 06.04.2026.

46. Building a Reliable Service for Sending Webhooks - Hookdeck Blog (практичні поради з побудови надійних outbound webhook-ів) – [Електронний ресурс]. – Режим доступу: <https://hookdeck.com/blog/building-reliable-outbound-webhooks> - Дата звернення: 12.04.2026.

47. Web View - Essential Kit (Unity asset) feature page - Voxel Busters Asset Store (опис WebView для Unity) – [Електронний ресурс]. - Режим доступу: <https://assetstore.essentialkit.voxelbusters.com/features/web-view> - Дата звернення: 21.02.2026.

48. Unity In-App Purchasing (IAP) - Unity Documentation (офіційний посібник з Unity IAP) – [Електронний ресурс]. – Режим доступу: <https://docs.unity.com/en-us/iap> - Дата звернення: 15.04.2026.

49. Privacy Policy - Vault (example privacy policy) - Vault (зразок політики конфіденційності) – [Електронний ресурс]. – Режим доступу: <https://vault.com/privacy-policy> - Дата звернення: 13.03.2026.

50. How Stripe built a game-changing app in a single flight with v0 - Vercel Blog (кейсове описання побудови застосунку з Vercel і Stripe) – [Електронний ресурс]. – Режим доступу: <https://vercel.com/blog/how-stripe-built-a-game-changing-app-in-a-single-flight-with-v0> - Дата звернення: 17.04.2026.

|      |      |          |        |      |                          |      |
|------|------|----------|--------|------|--------------------------|------|
|      |      |          |        |      | БР.ІІІ - 63.00.00.000 ПЗ | Арк. |
|      |      |          |        |      |                          | 73   |
| Змн. | Арк. | № докум. | Підпис | Дата |                          |      |

## **БІБЛІОГРАФІЧНА ДОВІДКА**

Тема бакалаврської роботи: “Розроблення інтелектуального веб-сервісу для монетизації та персоналізації ігрового досвіду”

Обсяг пояснювальної записки: 73 аркуші

Дата закінчення роботи червня 2026р.

Підпис \_\_\_\_\_