МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Веб-сервіс для продажу ігрових артефактів з використанням фреймворків React та Django**

Виконав: студент 4 курсу, групи 6.1210-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
О.В. Науменко
(ініціали та прізвище)
Керівник доцент каф. ЕІС та ПЗ, к.ф.-м.н., доцент
І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалавський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем_____
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна КРИТСЬКА
"_01_" __березня_____2024_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Науменку Олександру Володимировичу
(прізвище, ім'я, по батькові)

1.Тема роботи: **Веб-сервіс для продажу ігрових артефактів з використанням фреймворків React та Django**

керівник роботи ___Скрипник Ірина Анатоліївна, к.ф.-м.н., доцент_
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___26.12.2023 № 2215-с

2. Строк подання студентом кваліфікаційної роботи_____07.06.2024_____
3. Вихідні дані кваліфікаційної роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми створення веб-застосунків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- тестування програмної системи та розробка висновків та пропозицій.

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

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	01.03-10.03.24	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.03-12.03.24	виконано
3	Аналіз існуючих веб-сервісів для продажу ігрових артефактів	13.03-14.03.24	виконано
4	Вибір технологій для розробки веб-застосунку	15.03-20.03.24	виконано
5	Проектування архітектури веб-застосунку	21.03-26.03.24	виконано
6	Узгодження подальших дій з науковим керівником	27.03-28.03.24	виконано
7	Розробка бази даних для веб-застосунку	29.03-13.04.24	виконано
8	Розробка бекенду веб-застосунку з використанням Django	14.04-16.04.24	виконано
9	Розробка фронтенду веб-застосунку з використанням React	17.04-19.04.24	виконано
10	Інтеграція бекенду і фронтенду веб-застосунку	20.04-07.05.24	виконано
11	Тестування веб-застосунку	08.05-12.05.24	виконано
12	Реалізація користувацького інтерфейсу для веб-застосунку	13.05-20.05.24	виконано
13	Оформлення звіту	21.05-07.06.24	виконано

Студент _____ О.В. Науменко
(підпис) (прізвище та ініціали)

Керівник роботи _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 77

Рисунків — 19

Джерел — 13

Науменко О.В. Веб-сервіс для продажу ігрових артефактів з використанням фреймворків React та Django: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І.А. Скрипник. Запоріжжя : ЗНУ, 2024. 77 с.

У даній роботі представлено розробку та реалізацію веб-сервісу для продажу ігрових артефактів. Застосунок використовує Django та React для забезпечення інтерактивного та зручного інтерфейсу для користувачів, а також надійної backend-інфраструктури.

Основна мета роботи — створення інструменту, який дозволяє користувачам зручно купувати та продавати ігрові артефакти, тим самим забезпечуючи їм безпечну та ефективну платформу для обміну цифровими товарами. Використання сучасних веб-технологій дозволяє додатку надавати високий рівень користувацького досвіду та надійності.

Розробка веб-сервісу передбачає кілька ключових етапів: аналіз предметної області, огляд існуючих методів рішення, розробка архітектури системи та програмна реалізація. Особлива увага приділялася використанню сучасних фреймворків, таких як Django для серверної частини та React для клієнтської частини, для забезпечення швидкодії та масштабованості сервісу.

Ключові слова: *веб-сервіс, Django, React, ігрові артефакти, цифрові товари.*

ABSTRACT

Pages — 77

Figures — 19

Source — 13

Naumenko O.V. Web-Service for Selling Game Artefacts Using React and Django Frameworks: qualification work of the bachelor of specialty 121 "Software engineering" / science. manager I. A. Skrypnyk. Zaporizhzhia: ZNU, 2024. 77 p.

This work presents the development and implementation of a web service for the sale of game artifacts. The application uses Django and React to provide an interactive and user-friendly interface, as well as a reliable backend infrastructure.

The main goal of the work is to create a tool that allows users to conveniently buy and sell game artifacts, thereby providing them with a safe and efficient platform for exchanging digital goods. The use of modern web technologies allows the application to provide a high level of user experience and reliability.

Web service development involves several key stages: domain analysis, review of existing solution methods, system architecture development, and software implementation. Special attention was paid to the use of modern frameworks, such as Django for the server part and React for the client part, to ensure the speed and scalability of the service.

Keywords: *web service, Django, React, game artifacts, digital goods*.

ЗМІСТ

ВСТУП

В сучасному цифровому світі віртуальні ігри стають все більш популярними серед різних вікових та соціальних груп. Розвиток інтернет-технологій відкриває нові можливості для ігрової індустрії, зокрема для продажу ігрових артефактів, які впливають на геймплей та відчуття користувачів. У зв'язку з цим, актуальністю стає створення веб-сервісів для продажу ігрових артефактів, які забезпечать зручний та безпечний механізм для придбання віртуальних активів.

Мета дослідження

Метою цієї дипломної роботи є розробка та дослідження веб-сервісу для продажу ігрових артефактів з використанням фреймворків React та Django. Основною метою є створення ефективного та функціонального інтерфейсу, який задовольнятиме потреби користувачів у придбанні ігрових активів.

Завдання дослідження

- Розглянути теоретичні основи розробки веб-сервісів.
- Дослідити особливості фреймворків React та Django.
- Класифікувати ігрові артефакти та їх характеристики.
- Розробити веб-сервіс для продажу ігрових артефактів на базі фреймворків React та Django.
- Провести тестування та аналіз результатів.

Об'єкт дослідження

Об'єктом дослідження є процес розробки та функціонування веб-сервісу для продажу ігрових артефактів.

Предмет дослідження

Предметом дослідження є фреймворки React та Django, класифікація ігрових артефактів та їх особливості, а також практична реалізація веб-сервісу.

Методи дослідження

Для досягнення поставлених цілей будуть використані методи аналізу, класифікації, проектування та програмування, а також тестування та аналіз

результатів.

Практичне значення одержаних результатів

Одержані результати дослідження матимуть важливе практичне значення для розробників веб-сервісів, геймерської індустрії та широкого кола користувачів. Реалізація і аналіз веб-сервісу для продажу ігрових артефактів дозволить виявити переваги та недоліки використаних технологій та підходів, а також забезпечить підґрунтя для подальших досліджень у цій сфері.

РОЗДІЛ 1 ПОНЯТТЯ ВЕБ-СЕРВІСУ ТА ЙОГО ОСОБЛИВОСТІ

У сучасному світі інтернет-технології відіграють ключову роль у великій кількості сфер, включаючи торгівлю, розваги та інші. Один з найважливіших аспектів цього поля - розробка веб-сервісів, які є основним інструментом для взаємодії з користувачами через мережу Інтернет. У цьому розділі роботи будуть розглянуті теоретичні основи розробки веб-сервісу на прикладі створення сервісу для продажу ігрових артефактів з використанням фреймворків React та Django.

1.1 Поняття веб-сервісу та його особливості

Веб-сервіс - це програмне забезпечення, яке надає можливість взаємодії між різними програмами чи пристроями через мережу Інтернет. Це може бути функціональність, яка надається для взаємодії з базами даних, операцій з файлами, обміну даними між користувачами та іншими додатками. Веб-сервіси базуються на відкритих стандартах, таких як HTTP, XML та JSON, що робить їх доступними та інтероперабельними для різних систем.

Однією з ключових особливостей веб-сервісів є їхня спроможність працювати в розподілених системах. Це означає, що веб-сервіси можуть розташовуватися на різних серверах та працювати в різних мережевих середовищах, що дозволяє їм бути масштабованими та надійними.

Важливою характеристикою веб-сервісів є їхня безпека. За допомогою механізмів аутентифікації та авторизації веб-сервіси забезпечують захист від несанкціонованого доступу та зберігають конфіденційність даних, що передаються через мережу.

Іншою важливою особливістю веб-сервісів є їхня модульність та можливість інтеграції з існуючими системами. Це дозволяє створювати складні системи, які поєднують у собі різноманітні функціональність та даних з різних джерел.

Інтероперабельність є однією з ключових особливостей веб-сервісів. Це означає, що вони можуть працювати з різними технологіями та платформами. Це робить їх універсальними і дозволяє легко інтегрувати їх у вже існуючі інформаційні системи.

Ще однією важливою характеристикою є масштабованість. Веб-сервіси можуть бути масштабовані горизонтально (додаванням нових серверів) або вертикально (підвищенням ресурсів і потужності існуючих серверів) в залежності від потреб системи.

Наявність стандартизованих протоколів комунікації, таких як SOAP (Simple Object Access Protocol) або REST (Representational State Transfer), дозволяє забезпечити зручну взаємодію між клієнтами та серверами. REST є особливо популярним підходом, оскільки він базується на стандартних протоколах HTTP, що робить його простим у використанні та розумінні.

Важливо також зазначити, що веб-сервіси можуть бути публічними або приватними. Публічні веб-сервіси доступні для використання з будь-якої точки Інтернету, тоді як приватні веб-сервіси можуть бути обмежені лише для внутрішнього використання в межах організації чи компанії.

Загалом, веб-сервіси є важливою складовою інфраструктури Інтернету та сучасного програмного забезпечення. Вони надають зручний механізм для взаємодії між різними системами та додатками, сприяючи розширенню можливостей програмного забезпечення та розвитку нових технологій.

1.2 Архітектура веб-сервісів

Архітектура веб-сервісів - це спосіб організації програмного забезпечення для надання функцій через мережу Інтернет. Ця архітектура включає в себе різні компоненти, що взаємодіють між собою для обробки запитів, передачі даних та забезпечення функціональності. Ось деякі основні компоненти та концепції архітектури веб-сервісів:

1. Клієнт: Це програмне забезпечення або пристрій, який виконує

запити до веб-сервісу. Клієнти можуть бути веб-додатками, мобільними додатками, іншими веб-сервісами або будь-якими іншими системами.

2. Сервер: Це програмне забезпечення, яке надає функціональність або дані через мережу Інтернет. Сервер відповідає на запити від клієнтів, обробляє їх та повертає результати.

3. Протоколи комунікації: Веб-сервіси використовують різні протоколи комунікації для передачі даних між клієнтами та серверами. Найпоширенішими протоколами є HTTP (Hypertext Transfer Protocol), HTTPS (HTTP Secure), SOAP (Simple Object Access Protocol), REST (Representational State Transfer) та інші.’

4. Методи запитів: Веб-сервіси підтримують різні методи запитів для взаємодії з сервером. Найпоширеніші методи це GET (отримання даних), POST (створення даних), PUT (оновлення даних) та DELETE (видалення даних).

5. Формати даних: Дані, що передаються між клієнтами та серверами, зазвичай використовують стандартні формати, такі як XML (eXtensible Markup Language), JSON (JavaScript Object Notation), або YAML (YAML Ain't Markup Language).

6. Способи ідентифікації та автентифікації: Веб-сервіси можуть використовувати різні методи ідентифікації та автентифікації користувачів для забезпечення безпеки. Це може включати в себе використання токенів доступу, базову аутентифікацію або інші методи.

7. Безпека: Забезпечення безпеки веб-сервісів є важливою складовою архітектури. Це включає в себе захист від атак, шифрування даних, перевірку достовірності даних та інші заходи.

8. Моніторинг та аналіз: Для забезпечення ефективності та надійності веб-сервісів можуть використовуватися засоби моніторингу та аналізу, які дозволяють відстежувати роботу системи та вчасно виявляти проблеми.

Ці компоненти разом утворюють архітектуру веб-сервісів, яка дозволяє створювати потужні та ефективні системи для надання функціональності через мережу Інтернет.

1.3 Аналоги веб-сервісів для продажу ігрових артефактів

Розглянемо декілька аналогів таких сервісів для продажу:

На рисунку1.1 зображено наступне: розділ "Community Market" Steam, де відображено активні оголошення користувача про продаж ігрових предметів та баланс гаманця.

Рисунок *1.1 — Steam Community Market*

Steam Community Market є частиною платформи Steam, де гравці можуть купувати та продавати ігрові предмети, такі як скіни для зброї, картки колекційних ігор, та інші цифрові товари. Steam забезпечує безпеку транзакцій та має інтеграцію з ігровими акаунтами, що дозволяє автоматично передавати предмети між користувачами’.

Особливості:

- Інтеграція з великою кількістю ігор.
- Надійна система безпеки та захисту від шахрайства.
- Можливість встановлювати власні ціни на товари.
- Комісія за транзакції, яка стягується з кожного продажу.

На рисунку1.2 зображено наступне: сторінка з сайту g2g.com з пропозиціями по продажу предметів для гри Guild Wars 2 (EU), де відображені продавці, товари та їхні ціни.

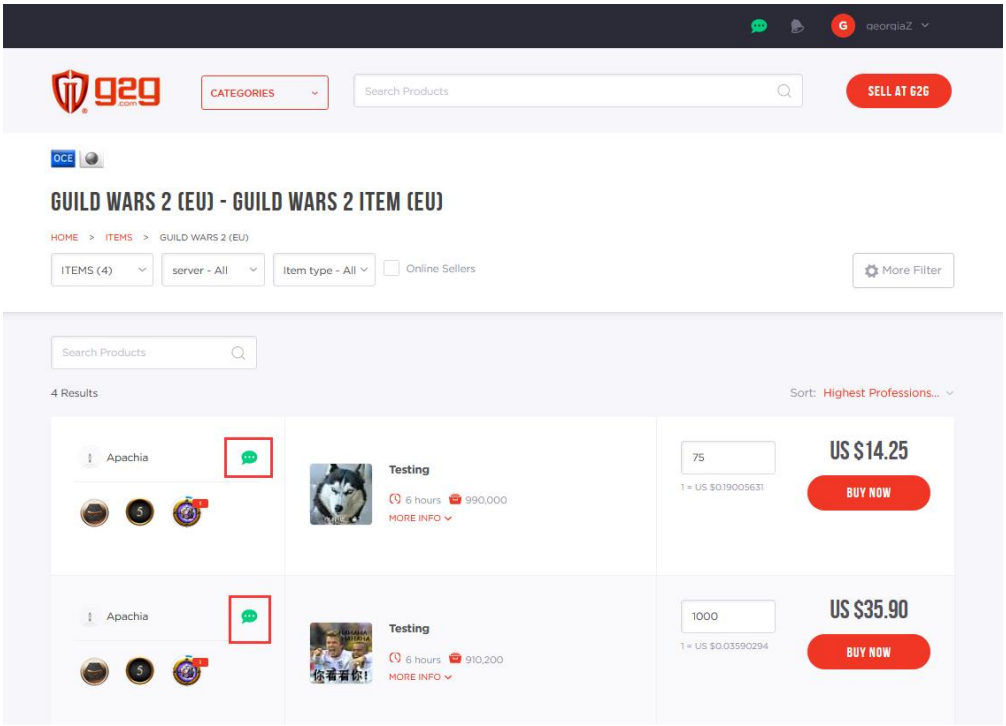

Рисунок *1.2 — G2G market*

Платформа для торгівлі ігровими товарами, яка дозволяє гравцям купувати та продавати ігрові акаунти, валюти, артефакти, та послуги з прокачування персонажів. Платформа забезпечує користувачам безпеку транзакцій та має систему відгуків для продавців і покупців.

Особливості:

- Широкий вибір ігрових товарів та послуг.
- Система рейтингів та відгуків для підвищення довіри.
- Захист транзакцій за допомогою системи Escrow.
- Різноманітні способи оплати.

На рисунку 1.3 зображено наступне: сторінка з сайту PlayerAuctions, де відображені деталі замовлень користувача на акаунти Genshin Impact з інформацією про ціни, статуси платежів та скасування замовлень.

Рисунок *1.3 — Player Auctions*

PlayerAuctions – це платформа для купівлі та продажу ігрових товарів та послуг. Вона включає в себе продаж ігрових акаунтів, валюти, предметів, та інших цифрових товарів. PlayerAuctions використовує систему Escrow для забезпечення безпечних транзакцій між користувачами.

Особливості:

- Велика кількість ігор та товарів.
- Система Escrow для захисту покупців і продавців.
- Підтримка різних способів оплати.
- Система рейтингів та відгуків.

На рисунку 1.4 зображено наступне: сторінка з сайту MMOGA з переліком ігрових предметів для подальшої купівлі на сайті

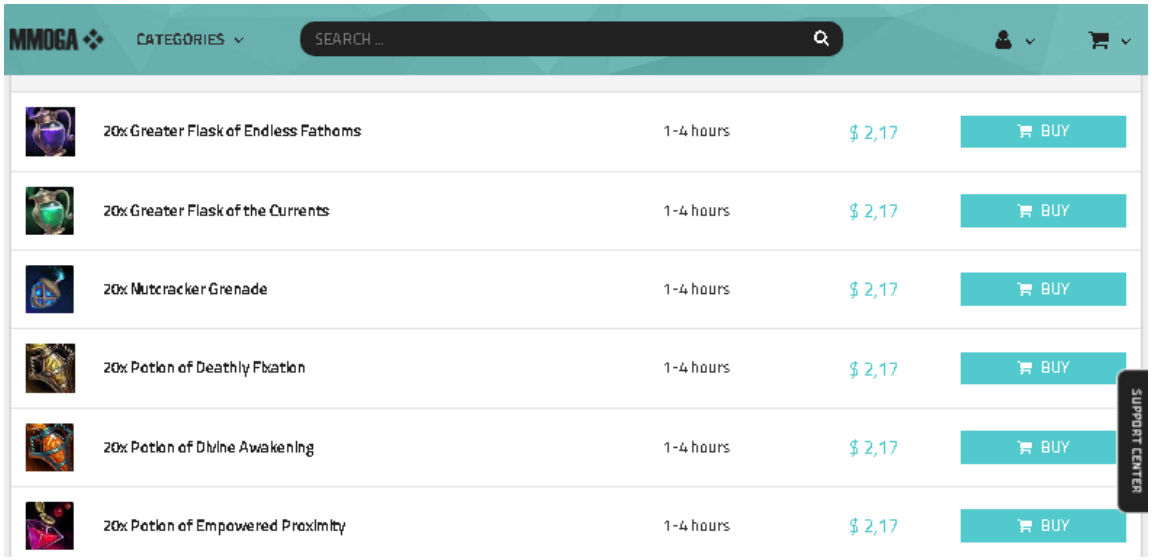

Рисунок *1.4 — MMOGA*

MMOGA є однією з провідних платформ для торгівлі ігровими товарами та послугами, яка пропонує купівлю і продаж ігрових валюти, предметів, акаунтів, та ключів до ігор. Платформа відома своєю швидкістю доставки товарів та високим рівнем безпеки.

Особливості:

- Швидка доставка ігрових товарів.
- Широкий асортимент ігрових товарів та послуг.
- Високий рівень безпеки транзакцій.
- Підтримка клієнтів 24/7.

На рисунку 1.5 зображено наступне: головна сторінка сайту Eldorado, де пропонується купівля та продаж ігрової валюти, предметів, акаунтів та послуг бустингу

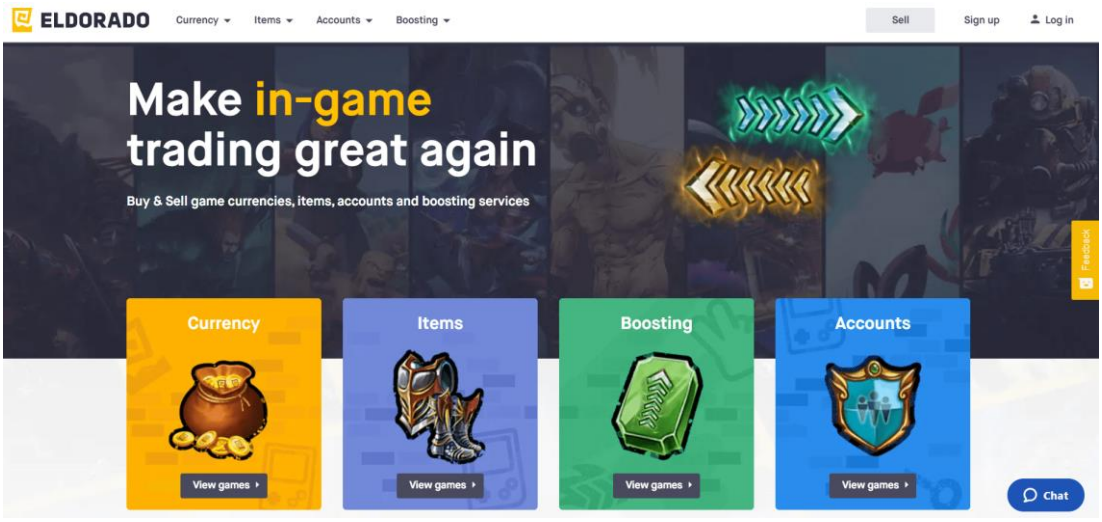

Рисунок *1.5 — Eldorado.gg*

Eldorado.gg є платформою для торгівлі ігровими акаунтами, предметами, валютою та іншими ігровими послугами.

Особливості:

- Платформа забезпечує захист транзакцій і захист покупців.
- Eldorado.gg гарантує швидку доставку товарів після покупки.
- Пропонує індивідуальні послуги для гравців, такі як бустинг.
- Підтримка клієнтів 24/7 забезпечує вирішення проблем у будь-який час.

На рисунку 1.6 зображено наступне: сторінка сайту OPSkins, з переліком різних ігрових артефактів на продаж

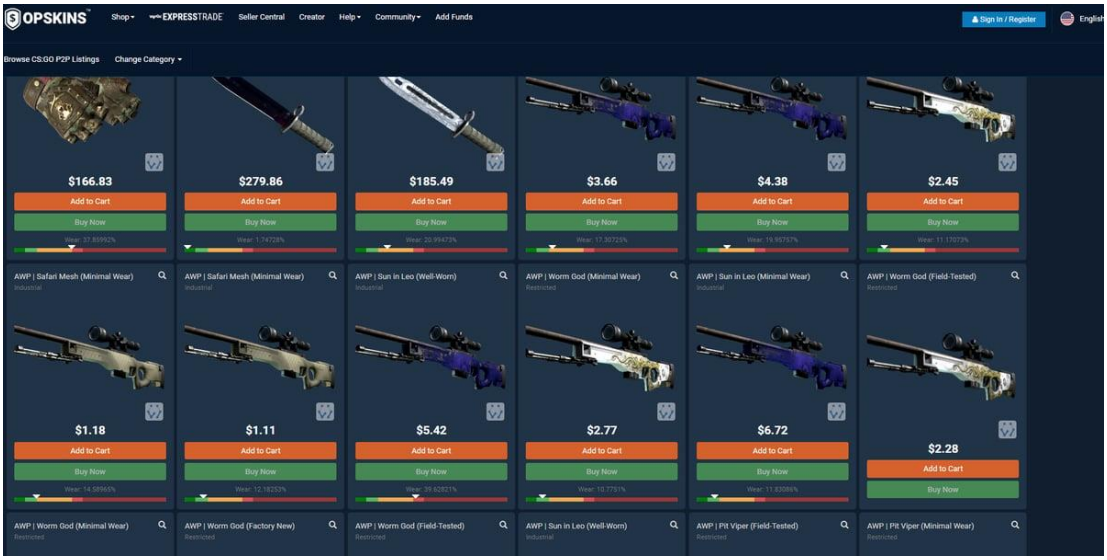

Рисунок *1.6 — OPSkins*

OPSkins спеціалізується на торгівлі внутрішньоігровими предметами, особливо для ігор, які мають скіни та інші косметичні предмети. Вона була популярною платформою до заборони від Valve на торгівлю з використанням їх API.

Особливості:

- Підтримка різних ігор та предметів.
- Проста та швидка система купівлі та продажу.
- Використання різних методів забезпечення безпеки транзакцій.
- Підтримка різних способів оплати, включаючи криптовалюти.

На рисунку 1.7 зображено наступне: сторінка сайту Dmarket, з переліком різних ігрових артефактів на продаж

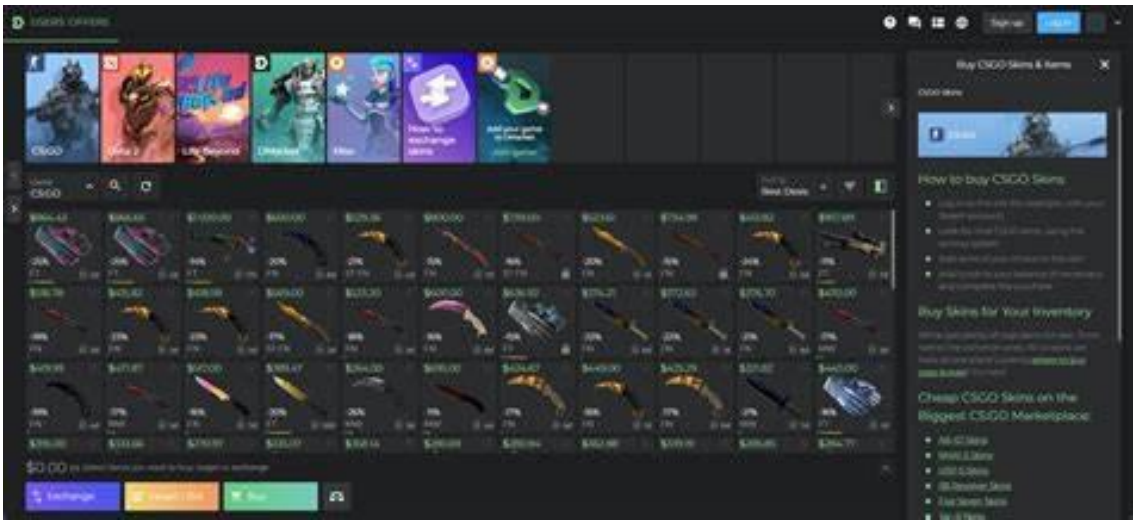

Рисунок *1.7 — Dmarket*

DMarket – це блокчейн-платформа для торгівлі внутрішньоігровими предметами. Вона дозволяє гравцям купувати, продавати та обмінювати цифрові товари на глобальному ринку.

Особливості:

- Використання технології блокчейн для забезпечення прозорості та безпеки транзакцій.
- Платформа надає доступ до світового ринку геймерів.
- Підтримка великої кількості популярних ігор.
- Можливість монетизувати внутрішньоігрові предмети.

На рисунку 1.8 зображено наступне: сторінка з сайту Gameflip, де відображені завершені транзакції з продажу цифрових товарів, таких як подарункові картки

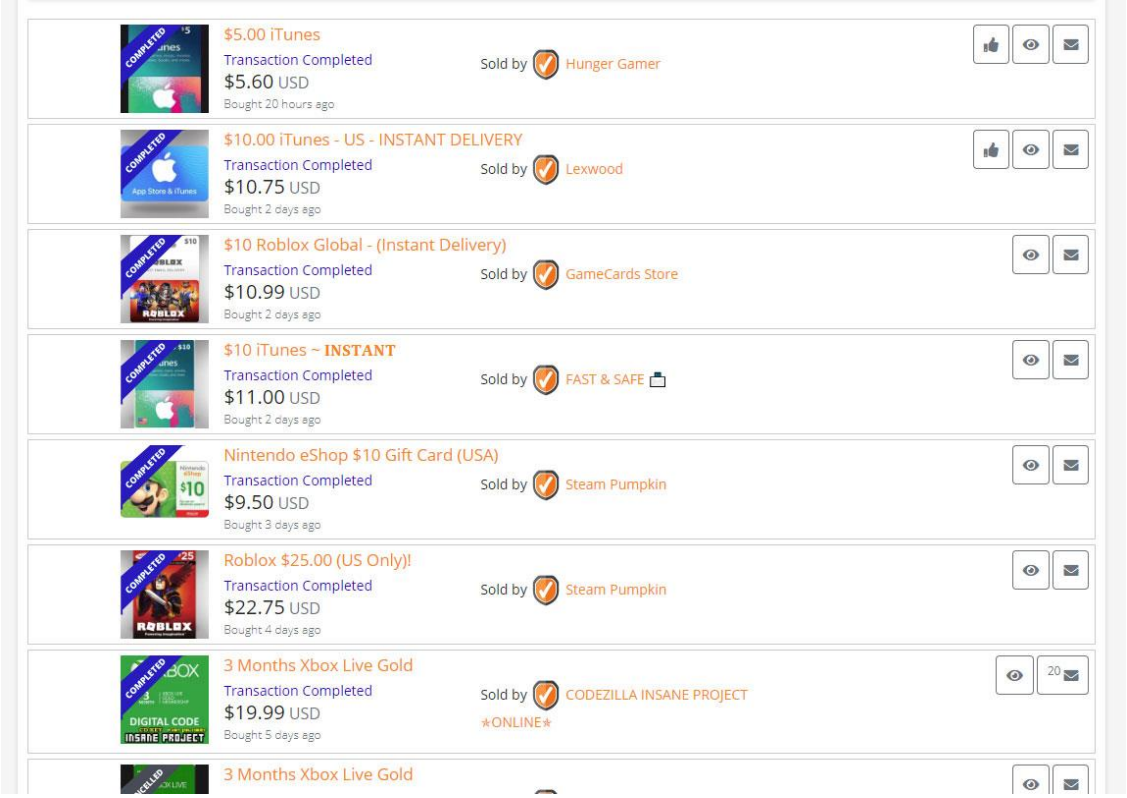

Рисунок *1.8 — Gameflip*

Gameflip є онлайн-ринком для купівлі та продажу ігрових предметів, внутрішньоігрової валюти, подарункових карток та акаунтів.

Особливості:

- Підтримка різних типів цифрових товарів, включаючи ігрові предмети та валюти.
- Програма захисту покупців забезпечує безпеку угод.
- Наявність мобільного додатку для зручності користування.
- Активна спільнота геймерів з можливістю обміну інформацією та відгуками.

На рисунку 1.9 зображено наступне: сторінка сайту Dmarket, з переліком різних ігрових артефактів на продаж

Рисунок *1.9 — Skinport*

Skinport є європейським онлайн-ринком для торгівлі скінами CS

та інших ігор, що надає можливість гравцям купувати та продавати внутрішньоігрові предмети з високим рівнем безпеки.

Особливості:

- Орієнтація на європейський ринок з підтримкою кількох мов.
- Використання двофакторної аутентифікації та інших методів безпеки.
- Чітка система комісій без прихованих витрат.
- Підтримка кредитних карток, банківських переказів та інших платіжних систем.

Проаналізувавши аналоги можна зрозуміти що вони мають такі особливості:

1) Безпека транзакцій: Усі платформи надають високий рівень безпеки, використовуючи системи Escrow або подібні механізми для захисту

покупців і продавців.

2) Різноманітність товарів: Всі перераховані платформи пропонують широкий асортимент ігрових товарів, включаючи акаунти, валюту, артефакти та інші цифрові предмети.

3) Системи відгуків та рейтингів: Більшість платформ мають системи рейтингів та відгуків, що дозволяють підвищити довіру користувачів до продавців.

4) Комісії за транзакції: Платформи стягують певну комісію за кожну транзакцію, що є стандартною практикою у цій галузі.

Після огляду аналогів і їх особливостей можна зробить висновки, що для створення конкурентоздатного веб-сервісу для продажу ігрових артефактів слід враховувати такі ключові аспекти, як безпека транзакцій, різноманітність товарів, системи відгуків і рейтингів, а також чітку політику комісій. Використання фреймворків React та Django дозволить створити гнучку та масштабовану платформу, яка зможе ефективно задовольнити потреби користувачів.

1.4 Технології розробки веб-сервісів

Існує безліч технологій для розробки веб-сервісів, кожна з яких має свої переваги та недоліки, а також відповідає на різні потреби розробників та проектів. Ось представлені найпопулярніших технологій для розробки веб-сервісів:

- Node.js: Node.js є середовищем виконання JavaScript на сервері, що дозволяє розробникам створювати ефективні та швидкодіючі веб-сервіси за допомогою JavaScript. Його асинхронний і подієвий характер дозволяє забезпечувати велику кількість одночасних з'єднань і роботу з великими обсягами даних.
- Spring Boot: Spring Boot є фреймворком для розробки веб-додатків

на мові програмування Java. Він надає розробникам широкий набір інструментів для створення різних типів додатків, включаючи веб-сервіси. Spring Boot дозволяє швидко підняти базовий веб-сервер та почати розробку з мінімальною конфігурацією.

- Django: Django є відкритим фреймворком для розробки веб-додатків на мові програмування Python. Цей фреймворк надає зручні засоби для створення веб-сервісів з використанням шаблонів, вбудованих модулів для роботи з базами даних та аутентифікації.
- ASP.NET Core: ASP.NET Core - це фреймворк для розробки веб-додатків на мові програмування C# в середовищі .NET Core. Він надає розробникам велику продуктивність та масштабованість, а також вбудовану підтримку для створення веб-сервісів.
- Ruby on Rails: Ruby on Rails - це фреймворк для розробки веб-додатків на мові програмування Ruby. Ruby on Rails надає конвенції над конфігурацією та зручні інструменти для створення RESTful веб-сервісів та інших веб-додатків.

Ці технології представляють лише деякі з варіантів для розробки веб-сервісів. Вибір конкретної технології залежить від потреб вашого проекту, ваших вмінь та вимог вашої команди.

1.5 Фреймворк React: опис та можливості

React - це відкритий JavaScript фреймворк, розроблений компанією Facebook, який використовується для створення користувацьких інтерфейсів. Він заснований на компонентній архітектурі, що дозволяє розбити інтерфейс на невеликі, незалежні компоненти, що полегшує розробку та підтримку коду. Однією з ключових особливостей React є використання віртуального DOM, який забезпечує оптимізацію продуктивності та ефективне оновлення сторінок без перезавантаження сторінки.

Для опису користувацького інтерфейсу React використовує JSX - це розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код безпосередньо в JavaScript. Це полегшує створення декларативного та легкочитаного коду. React також ідеально підходить для розробки односторінкових додатків (SPA), де зміни в контенті сторінки відбуваються без перезавантаження сторінки, що забезпечує зручний користувацький досвід.

Завдяки своїм можливостям для реактивного оновлення інтерфейсу та широкій екосистемі бібліотек та інструментів, React став одним з найпопулярніших фреймворків для розробки веб-додатків. Він також може використовуватися для створення мобільних додатків за допомогою фреймворку React Native, що робить його універсальним інструментом для розробки веб- та мобільних додатків.

React має кілька переваг, які зробили його одним з найпопулярніших фреймворків для розробки веб-додатків.

Переваги React:

1. Компонентна архітектура: React базується на компонентній архітектурі, що дозволяє розбивати інтерфейс на невеликі, незалежні компоненти. Це спрощує розробку, тестування та підтримку коду.

2. Віртуальний DOM: Використання віртуального DOM дозволяє React оптимізувати продуктивність та ефективність оновлення сторінок. Він оновлює лише ті елементи сторінки, які змінилися, що робить додатки швидшими та ефективнішими.

3. Широка екосистема: React має велику та активну екосистему, включаючи багато корисних бібліотек, інструментів та розширень, які допомагають в розробці різних типів додатків.

4. JSX: JSX дозволяє писати HTML-подібний код безпосередньо в JavaScript, що полегшує створення декларативного та легкочитаного коду.

5. Підтримка односторінкових додатків (SPA): React ідеально підходить для розробки односторінкових додатків, де сторінка не

перезавантажується при зміні контенту, що забезпечує зручний користувацький досвід.

Незважаючи на свої переваги, React має деякі недоліки:

1. Велика кількість вивчення: React може виявитися складним для новачків, особливо для тих, хто не має досвіду з компонентною архітектурою або віртуальним DOM.

2. Вимоги до екосистеми: Використання деяких додаткових бібліотек або інструментів, які зазвичай використовуються разом з React, може вимагати додаткового часу для їх вивчення та інтеграції.

3. Підтримка оновлень: Періодичні оновлення та зміни в екосистемі React можуть призвести до необхідності оновлення коду додатків та внесення змін у вже наявний функціонал.

Незважаючи на ці недоліки, React залишається одним з найпопулярніших та потужних фреймворків для розробки веб-додатків з великим спільнотою користувачів та активним розвитком.

1.6 Фреймворк Django: опис та можливості

Django - це відкритий веб-фреймворк, що розробляється на мові програмування Python. Він володіє великою популярністю та широким спектром можливостей для швидкої та ефективної розробки веб-додатків. Однією з головних переваг Django є його заснованість на принципах "кодування менше, робіть більше", що дозволяє розробникам прискорити процес розробки за рахунок використання готових компонентів та шаблонів.

Фреймворк Django пропонує вбудовані інструменти та модулі, які спрощують роботу з базами даних, маршрутизацією URL, автентифікацією, безпекою та іншими аспектами веб-розробки. Використання цих готових рішень дозволяє розробникам зосередитися на бізнес-логіці своїх додатків, не

витрачаючи час на написання рутинного коду.

Крім того, Django має вбудований адміністративний інтерфейс, який дозволяє адміністраторам легко керувати даними веб-додатків, використовуючи зручний та інтуїтивно зрозумілий інтерфейс. Це важлива функціональність для швидкого впровадження та підтримки веб-проектів, особливо на ранніх етапах їх розвитку.

Загалом, Django володіє широкими можливостями для створення високоякісних та масштабованих веб-додатків, і його використання спрощує процес розробки, дозволяючи розробникам швидше та ефективніше створювати веб-застосунки різного рівня складності.

1.7 Ігрові артефакти: класифікація та особливості

Ігрові артефакти є ключовим елементом багатьох веб-сервісів, спрямованих на геймерську аудиторію. Класифікація ігрових артефактів може бути здійснена за різними критеріями, такими як їх функціональне призначення, цінова категорія або тип гри.

По функціональному призначенню ігрові артефакти можна розділити на кілька категорій, таких як предмети екіпірування (зброя, броня, аксесуари), валюта гри (золото, кристали, гроші), ігрові ресурси (матеріали для крафту, енергія, досвід), а також різноманітні предмети для прикрашання або персоналізації персонажів.

Особливості ігрових артефактів включають їх важливість для геймплею, рівень рідкості, способи отримання (чи то через геймплей, чи платні транзакції), а також їх вплив на ігрову економіку та баланс гри. Більшість ігрових артефактів мають свої унікальні характеристики та ефекти, які можуть змінювати геймплей та взаємодію гравців у віртуальному світі.

Окрім цього, ігрові артефакти можуть мати різний рівень впливу на ігровий процес. Деякі з них можуть бути критично важливими для досягнення успіху в грі або впливати на результати битв чи сюжетні лінії. Інші можуть

бути лише декоративними або надавати додатковий естетичний сприйняття гри, не впливаючи на головний геймплей.

Успішний веб-сервіс для продажу ігрових артефактів має враховувати ці особливості та класифікацію, щоб надати гравцям максимально зручний та цікавий досвід. Це може включати в себе створення ефективної системи фільтрації та сортування товарів, забезпечення належної інформації про кожен артефакт, а також розробку механізмів для підтримки грифічної привабливості та інтерактивності веб-інтерфейсу. Крім того, важливо забезпечити безпеку та надійність операцій з покупками, враховуючи вимоги захисту персональних даних і фінансових операцій гравців.

1.8 Постановка задачі

Мета кваліфікаційної роботи – створення веб-сервісу для продажу ігрових артефактів з використанням фреймворків React та Django. Впровадження сучасних веб-технологій у сферу продажу цифрових товарів має на меті полегшити організацію роботи з клієнтами, забезпечити безпеку транзакцій та покращити користувацький досвід.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

Використовуючи СУБД (систему управління базами даних) PostgreSQL, створити спроектовану базу даних на SQL.

Спроектувати та реалізувати структуру бази даних для зберігання інформації про ігрові артефакти, користувачів, транзакції та інші необхідні дані.

Виконати на SQL необхідні дії з додавання, модифікації і видалення даних у таблицях створеної бази даних.

Розробити серверну частину веб-сервісу на фреймворку Django.

Створити API для взаємодії з базою даних та забезпечення функціональності сервісу.

Реалізувати бізнес-логіку для обробки запитів користувачів та

проведення транзакцій.

Розробити клієнтську частину веб-сервісу на фреймворку React.

Створити інтуїтивно зрозумілий і зручний інтерфейс користувача для взаємодії з сервісом.

Забезпечити динамічну взаємодію клієнтської частини з серверною частиною через API.

Реалізувати захищені з'єднання між клієнтською та серверною частинами за допомогою SSL/TLS.

Провести тестування розробленого веб-сервісу для забезпечення його надійності та коректності роботи.

Виконати юніт-тести, інтеграційні тести та функціональні тести.

1.9 Висновок до розділу 1

У результаті проведеного дослідження, огляду аналогів і аналізу теоретичних основ розробки веб-сервісу для продажу ігрових артефактів з використанням фреймворків React та Django, я, як виконавець дипломної роботи, прийшов до висновку про важливість глибокого розуміння технічних аспектів веб-розробки та особливостей геймінгу.

Для успішної реалізації проекту необхідно використовувати передові технології, такі як React і Django, і приділяти увагу деталям у класифікації та представленні ігрових артефактів. Крім того, важливо враховувати потреби та вимоги користувачів, забезпечуючи їм безпечну та зручну платформу для придбання ігрових активів.

Відтак, результати цього дослідження відкривають нові перспективи для подальшого розвитку веб-сервісів у галузі геймінгу та створення інноваційних рішень для задоволення потреб сучасних гравців.

РОЗДІЛ 2 ТЕХНОЛОГІЯ РОЗРОБКИ ВЕБ-СЕРСВІСУ

Перед початком будь-якої розробки важливо ретельно проаналізувати потреби та очікування користувачів. Це може включати вивчення їхніх вимог до функціональності, зручності використання та інших аспектів. Наприклад, користувачі можуть очікувати зручного інтерфейсу для перегляду та покупки ігрових артефактів, системи реєстрації та авторизації, можливості відстеження замовлень тощо.

2.1 Вибір технологій

На цьому етапі зосередився на виборі технологій, які будуть використовуватися в моєму проекті. Провів дослідження і вивчив різні опції для фронтенду, бекенду та бази даних з метою забезпечення ефективності та продуктивності розробки.

У зв'язку з цим були зроблені такі вибори:

Щодо фронтенду, я вирішив використовувати фреймворк React, оскільки він забезпечує швидку розробку інтерактивних інтерфейсів з використанням компонентної архітектури. Також розглядаю використання Redux для керування станом додатку, що допоможе підтримувати чистоту та прозорість коду.

Щодо бекенду, вибором став фреймворк Django на мові програмування Python. Django забезпечує готові рішення для багатьох аспектів веб-розробки, таких як автентифікація, маршрутизація та робота з базою даних. Для створення API я розглядаю використання Django REST Framework, оскільки він надає зручні інструменти для створення RESTful API.

Щодо бази даних, розглядаю використання для розробки та тестування, оскільки вона є легкою в налаштуванні та інтеграції з Django. Проте для продакшн-середовища планую використовувати PostgreSQL, яка є потужною та надійною реляційною базою даних.

Для забезпечення ефективності розробки та розгортання, також розглядаю використання різних інструментів, таких як Webpack, Babel, NPM або Yarn, Git та Docker.

Ці вибори зроблені з метою створення потужного, ефективного та безпечного веб-сервісу для продажу ігрових артефактів, який задовольнятиме потреби користувачів і відповідатиме сучасним стандартам розробки.

2.2 Проектування бази даних

На рисунку 2.2 зображено логотип MySQL.

Рисунок *2.1 — логотип MySQL*

Аналіз вимог:

Перед тим, як приступити до створення бази даних, я провів детальний аналіз вимог щодо зберігання даних. Це включало вивчення різних сутностей, які повинні були б представлені в базі даних, та їх взаємозв'язків. Наприклад, я розглядав сутності, такі як користувачі, продукти (ігрові артефакти), замовлення тощо.

Структурування даних:

На основі вимог і аналізу я створив структуру бази даних. Це включало визначення таблиць та полів для кожної сутності, а також визначення зв'язків

між ними. Наприклад, для таблиці користувачів я визначив поля для зберігання інформації про ім'я, електронну пошту, пароль тощо.

Вибір типів даних та обмежень:

На цьому етапі я обрав відповідні типи даних для кожного поля, а також встановив обмеження і правила для даних. Наприклад, для поля "ім'я" я використав тип VARCHAR з обмеженням на кількість символів, а для поля "електронна пошта" - тип EMAIL з обов'язковим унікальним індексом.

Нормалізація бази даних:

Я використав принципи нормалізації для зменшення дублювання даних та забезпечення цілісності даних. Це допомагає уникнути аномалій та забезпечити ефективне зберігання даних.

Оптимізація для продуктивності:

На останньому етапі я провів оптимізацію бази даних для досягнення максимальної продуктивності. Це включало індексацію ключових полів, використання кешування та інші стратегії для підвищення швидкодії операцій з даними.

2.3 Розробка фронтенду з використанням React

На рисунку 2.2 зображено наступне: логотип React. Від логотипу відходять стрілки до шести ключових характеристик React, кожна з яких супроводжується іконкою та підписом:

- **Ease of Use** (Іконка книги) – Легкість у використанні.
- **Seamless Update** (Іконка оновлення) – Безперервне оновлення.
- **Developer Tools** (Іконка інструментів) – Інструменти для розробників.
- **The Virtual DOM** (Іконка монітора) – Віртуальний DOM.
- **Reusable Components** (Іконка компонентів) – Повторно використовувані компоненти.
- **Opensource** (Іконка глобуса) – Відкритий вихідний код.

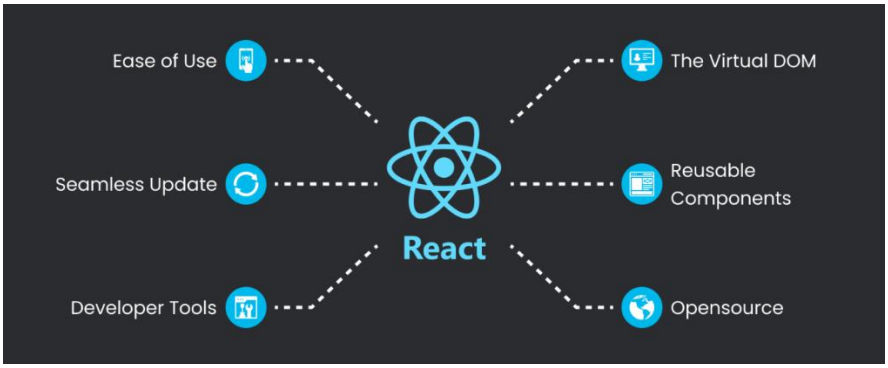

Рисунок *2.2* — *React*

Фронт-енд веб-сервісу буде поділено на три ключові компоненти:

1. Header: Цей елемент буде служити візитною карткою сайту, розміщуючи логотип, панель навігації та кнопку авторизації. Він буде нести відповідальність за перше враження користувачів, надаючи їм доступ до основних функцій та контенту веб-сервісу.

2. Main: Як серце сайту, Main буде динамічно відображати різноманітний контент, залежно від потреб користувача. Тут буде представлений список ігрових артефактів, детальна інформація про них, форма для додавання нових артефактів, кошик для покупок та інші функціональні блоки. Цей компонент стане центром взаємодії користувачів з веб-сервісом, надаючи їм все необхідне для пошуку, покупки та управління ігровими артефактами.

3. Footer: Завершуючи композицію сайту, Footer буде містити контактну інформацію та посилання на інші сторінки. Він стане зручним довідником для користувачів, які шукають додаткову інформацію або потребують допомоги.

Кожен компонент буде ретельно розроблений як окремий React-компонент, що володіє власною логікою та функціональністю. Цей підхід забезпечить модульність, гнучкість та простоту обслуговування коду.

Функціональні можливості компонентів:

- Відображення UI: Кожен компонент буде відповідальним за візуальне представлення своєї частини інтерфейсу, використовуючи React та Material UI.
- Обробка користувацького вводу: Компоненти будуть реагувати на дії користувачів, такі як натискання кнопок, введення даних, прокручування сторінки та інші.
- Взаємодія з back-end API: За допомогою Axios компоненти будуть отримувати дані, оновлювати інформацію та надсилати запити до back-end API, забезпечуючи динамічну роботу веб-сервісу.

Для гарантування безперебійної роботи та високої якості фронтенду буде використовуватися Jest. Цей фреймворк дозволить писати модульні тести для компонентів React, а також тести для API та інших аспектів фронтенду, забезпечуючи його стійкість до помилок та відповідність вимогам.

Також буде використовуватися Material UI для створення UI-компонентів, що відповідають Material Design. Цей дизайн-система Google пропонує чіткі, зрозумілі та естетично приємні UI-елементи, що робить інтерфейс веб-сервісу зручним, інтуїтивно зрозумілим та привабливим для користувачів.

Для маршрутизації між різними сторінками та URL-адресами веб-сервісу, буде використовуватися React Router. Цей фреймворк дозволить створювати односторінкові веб-додатки (SPA), які не перезавантажують сторінку при переході між різними розділами, забезпечуючи плавний та швидкий досвід користування.

Щоб керувати станом front-end-додатку, буде використаний Redux. Цей фреймворк дозволить зберігати дані про стан в єдиному місці, роблячи їх доступними для всіх компонентів. Це забезпечить централізоване управління, кращу передбачуваність та полегшить синхронізацію даних між різними частинами веб-сервісу.

Використання Axios: Axios буде використовуватися для надсилання HTTP-запитів до back-end API. Ця бібліотека полегшує роботу з API, роблячи код більш чітким, читабельним та зручним для обслуговування.

Використання React-Bootstrap: React-Bootstrap буде використовуватися для створення UI-компонентів, заснованих на Bootstrap. Цей CSS-фреймворк пропонує широкий спектр готових UI-елементів, таких як кнопки, форми, модальні вікна, сітки та інші. React-Bootstrap дозволить нам швидко та ефективно розробити привабливий та функціональний інтерфейс, не витрачаючи часу на написання стилів з нуля.

Інструменти розробки: Окрім бібліотек, ми будемо використовувати сучасні інструменти розробки для підвищення продуктивності та ефективності. Веб-пак (Webpack) буде використовуватися для збирання та оптимізації коду JavaScript, CSS та інших ресурсів. Babel дозволить нам використовувати сучасні функції JavaScript, транспілюючи їх у код, сумісний зі старішими браузерами.

Після завершення розробки фронтенд буде розгорнуто на відповідному хостингу. Веб-сервер, такий як Nginx або Apache, буде відповідальним за обслуговування запитів користувачів та надсилання їм статичних файлів фронтенду.

2.4 Розробка бекенду з використанням Django

На рисунку 2.3 зображено наступне: логотип Django .

Розробка бекенду з використанням Django була ключовим етапом у процесі створення веб-сервісу. Почавши розробку, я налаштував середовище, встановивши Python та Django. Використання віртуального середовища дозволило мені уникнути конфліктів версій пакетів та забезпечити ізольоване середовище для розробки.

Рисунок *2.3 — логотип Django*

Для створення основної структури проекту використав команду django-admin startproject. Налаштувавши основні параметри в файлі settings.py, такі як налаштування бази даних та шляхи до статичних та медіафайлів, я забезпечив базову конфігурацію проекту.

Для різних частин функціональності мого веб-сервісу я створив окремі додатки. Кожен додаток був відповідальний за певну частину функціональності, що спростило структуру проекту та його підтримку.

З використанням ORM Django я створив моделі даних для представлення сутностей мого веб-сервісу. Визначивши зв'язки між моделями, я виконав міграції для створення таблиць в базі даних та забезпечив консистентність даних.

На цьому етапі була написана велика кількість функцій та методів, які виконували різноманітні операції з даними. Використання Django Forms та Django REST Framework допомогло в обробці та валідації даних, які надходили від клієнтів.

Одним із пріоритетів було забезпечення безпеки мого веб-сервісу. За допомогою вбудованих засобів Django я налаштував механізми захисту від різноманітних атак, таких як Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF).

Провівши тестування, я переконався в коректному функціонуванні мого бекенду. Написавши юніт-тести та інтеграційні тести, я перевірив різні аспекти роботи мого коду.

Розробка бекенду з використанням Django дозволила створити надійний та масштабований сервіс, забезпечивши його ефективну роботу та безпеку.

2.5 Інтеграція фронтенду і бекенду

Після успішної розробки фронтенду та бекенду, наступним важливим етапом була їхня інтеграція. Ось як я виконав цей процес:

1. Налаштування взаємодії: Перш ніж розпочати інтеграцію, я визначив API-методи, які будуть використовуватися фронтендом для отримання та надсилання даних. Це включало створення URL-шляхів, які відповідали за виклик певних функціональностей бекенду.

2. З'єднання фронтенду і бекенду: Для забезпечення взаємодії фронтенду та бекенду, я використовував HTTP-запити, такі як GET, POST, PUT та DELETE. Це дозволило фронтенду взаємодіяти з різними ендпоінтами бекенду для отримання та оновлення даних.

3. Обробка даних: Після виклику API-методів з фронтенду, бекенд обробляв ці запити, виконуючи відповідні операції з базою даних або з іншими системами. Результати були повернуті фронтенду у вигляді відповідей на запити.

4. Валідація та обробка помилок: Важливою частиною інтеграції була обробка помилок. Я використовував стандартні HTTP-статуси помилок та повідомлення про помилки для передачі інформації про помилки з бекенду на фронтенд. Це дозволило коректно обробляти помилки та повідомляти користувачам про них.

5. Тестування: Після інтеграції важливо було перевірити правильність взаємодії між фронтендом та бекендом. Я проводив інтеграційне тестування, щоб переконатися, що всі API-методи працюють правильно та

взаємодіють між собою без проблем.

Інтеграція фронтенду та бекенду була важливим етапом у розробці веб-додатку. Цей процес дозволив створити повнофункціональний та надійний додаток, який ефективно взаємодіяв з користувачем та забезпечував йому потрібну функціональність.

2.6 Тестування і налагодження

Після успішної інтеграції фронтенду та бекенду настала черга тестування та налагодження системи, щоб переконатися в її стабільності та відповідності функціональним вимогам. Процес тестування та налагодження був важливим для забезпечення високої якості продукту та коректної роботи всіх його компонентів.

У рамках тестування використовувалися різні стратегії та методи, щоб виявити та усунути можливі проблеми:

Юніт-тести були написані для перевірки окремих компонентів системи, таких як функції, методи та класи, без залежностей від інших частин програми. Це дозволило переконатися в коректності їх роботи та відсутності помилок у вузькому контексті.

Інтеграційне тестування включало перевірку взаємодії різних компонентів системи. Зокрема, тести перевіряли відповідність поведінки системи відповідно до заданих специфікацій та правильність передачі даних між фронтендом та бекендом через API.

Валідація даних відігравала важливу роль у перевірці коректності обробки та валідації вхідних даних системи. Переконавшись, що система правильно обробляє некоректні вхідні дані та видає відповідні повідомлення про помилки, можна було забезпечити зручне користування для кінцевого користувача.

Під час налагодження (debugging) були виявлені та виправлені різні помилки та проблеми, що виникали під час розробки. Використання різних інструментів відлагодження, таких як виведення на консоль, логування та інші, допомогло виявити та усунути недоліки швидко та ефективно.

Створення тестових середовищ для випробування додатку у різних умовах було корисним, оскільки це дозволило виявити проблеми та несправності, які можуть виникати під час реальної експлуатації програмного забезпечення.

У підсумку, тестування та налагодження були ключовими кроками у забезпеченні якості та надійності розробленого продукту, а також у забезпеченні коректної взаємодії між фронтендом та бекендом.

2.7 Висновок за розділом 2

В цьому розділі було виконано проектуванню та розробка веб-сервісу для продажу ігрових артефактів з використанням фреймворків React та Django, був ключовим у створенні функціонального та ефективного продукту. Під час проектування були визначені основні вимоги до системи, спроектована її архітектура та розроблені основні компоненти. Використання фреймворків React та Django надало можливість швидкого та ефективного розробляти фронтенд та бекенд, відповідно.

Під час розробки фронтенду з використанням React я був здатен створити користувацький інтерфейс, який був інтуїтивно зрозумілий та зручний у використанні для користувачів. Використання компонентного підходу дозволило ефективно керувати структурою проекту та підтримувати його.

У свою чергу, розробка бекенду з використанням Django дозволила створити надійний та масштабований серверний компонент системи. Використання ORM Django дозволило швидко створити та взаємодіяти з

базою даних, а також розробити API для взаємодії з фронтендом.

Після успішної розробки фронтенду та бекенду була виконана їхня інтеграція.

Цей розділ є фундаментальним для успішної розробки та реалізації проекту, надаючи чітке розуміння вимог до системи та розроблюючи ефективні механізми для їх втілення.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-СЕРВІСУ

3.1 Розробка фронт-енду

Фронт-енд веб-сервісу буде розроблено за допомогою фреймворку React. React — це JavaScript-бібліотека для створення динамічних інтерфейсів користувача. Вона використовує віртуальне DOM (Virtual Document Object Model) для оптимізації оновлення інтерфейсу. Це робить React ідеальним вибором для розробки веб-додатків з високою продуктивністю та масштабованістю.

Основні компоненти фронт-енду:

- Головна сторінка: Ця сторінка буде відображати список доступних ігрових артефактів. Користувачі зможуть переглядати інформацію про артефакти, такі як назва, опис, ціна та зображення.
- Сторінка артефакту: Ця сторінка буде відображати детальну інформацію про конкретний артефакт. Користувачі зможуть переглянути всі характеристики артефакту, а також купити його.
- Кошик: Ця сторінка буде відображати список артефактів, які користувач додав до кошика. Користувачі зможуть змінити кількість артефактів у кошику, а також оформити замовлення.
- Сторінка оформлення замовлення: Ця сторінка буде збирати інформацію про користувача, необхідну для оформлення замовлення, таку як ім'я, адреса та номер телефону.

3.1.1 Розгляд основних частин фронт-енду

Цей компонент відповідає за логін користувача через Steam. При успішному вході користувач перенаправляється на головну сторінку.

Лістинг *1 - частина коду файла SteamLoginPage.jsx:*

```
export default function SteamLoginPage({ redirectTo }) {
  const { data, setData } = useAppContext();
  const params = new
URLSearchParams(window.location.search);

  const [status, setStatus] = useState(null);

  useEffect(() => {
    (async () => {
      try {
        const data = await fetch(
          `http://localhost:8000/api/v1/auth?assoc_handle=$
{params.get(
            "openid.assoc_handle"
          )}&claimed_id=${params.get(
            "openid.claimed_id"
          )}&identity=${params.get(
            "openid.identity"
          )}&response_nonce=${params.get(
            "openid.response_nonce"
          )}&return_to=${params.get("openid.return_to")}&si
g=${params.get(
            "openid.sig"
          )}&signed=${params.get("openid.signed")}`,
          { method: "GET", headers: {} }
        );
        if (data.ok) {
          const response = await data.json();
          setData({ user: response });
          setStatus({ code: 200, message: "OK" });
        } else setStatus({ code: 403, message: "Forbidden"
});
      } catch (error) {
        setStatus({ code: 503, message: "Service
Unavailable" });
      }
    })();
  }, []);
```

```
  if (status == null) {
    return (
      <div className="spinner-container" style={{
marginTop: "200px" }}>
        <Spinner animation="border" role="status">
          <span className="visually-
hidden">Loading...</span>
        </Spinner>
      </div>
    );
  } else if (status["code"] == 200) window.location.href =
"/";
  else return <ErrorPage errorInfo={status} />;
}
```

Опис його функціональності:

- Компонент використовує хуки useEffect та useState для керування станом та виконання побічних ефектів.
- Під час першого рендеру виконується асинхронний запит до сервера для перевірки логіну користувача через Steam.
- Використовується метод fetch з HTTP методом GET та відповідними заголовками, що містять токен користувача і його Steam ID.
- Відповідь сервера обробляється для перевірки статусу: якщо відповідь успішна (статус 200), стан оновлюється, і користувач перенаправляється на головну сторінку.
- У випадку помилки, компонент показує відповідну сторінку помилки.
- Компонент також відображає анімацію завантаження, поки очікується відповідь від сервера.

Цей компонент відображає профіль користувача, включаючи його інвентар та інформацію про профіль.

Лістинг *2 - частина коду файла ProfilePage.jsx:*

```
export default function ProfilePage() {
  const { data, setData } = useAppContext();
```

```
  const [status, setStatus] = useState(null);
  const [items, setItems] = useState([]);
  const [popup, setPopup] = useState(false);
  const [popupItem, setPopupItem] = useState(null);

  const [searchTerm, setSearchTerm] = useState("");
  const [currentPage, setCurrentPage] = useState(1);
  const itemsPerPage = 12;
  const [totalItems, setTotalItems] = useState(0);

  const [sortOption, setSortOption] = useState("");

  useEffect(() => {
    async function fetchInventory() {
      if (data && data.user && data.user.token &&
data.user.steam_id) {
        try {
          const response = await fetch(
            `http://127.0.0.1:8000/api/v1/get_inventory/?se
arch=${searchTerm}&page=${currentPage}&limit=${itemsPerPage
}&sort_by=${
              sortOption.split("-")[0]
            }&order=${sortOption.split("-")[1]}`,
            {
              method: "GET",
              headers: {
                token: data.user.token,
                steam: data.user.steam_id,
              },
            }
          );

          if (response.ok) {
            const result = await response.json();
            setTotalItems(result.total_items);
            setCurrentPage(result.page);
            setItems(result.descriptions);
            setStatus({ code: 200, message: "OK" });
          } else {
            setStatus({ code: response.status, message:
response.statusText });
          }
        } catch (error) {
          setStatus({ code: 503, message: "Service
Unavailable" });
        }
      }
    }
```

```
  fetchInventory();
}, [data, searchTerm, currentPage, sortOption]);

const handleSearchChange = (e) => {
  setSearchTerm(e.target.value);
  setCurrentPage(1);
};

const handlePageChange = (newPage) => {
  setCurrentPage(newPage);
};

const handleItemClick = (item) => {
  setPopupItem(item);
  setPopup(true);
};

const closePopup = () => {
  setPopupItem(null);
  setPopup(false);
};

const handleSortChange = (option) => {
  setSortOption(option);
};

const handleSellItem = async () => {
  if (data && data.user && popupItem) {
    try {
      const response = await fetch(
        `http://127.0.0.1:8000/api/v1/sell_item/`,
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            token: data.user.token,
            steam: data.user.steam_id,
          },
          body: JSON.stringify({
            item_id: popupItem.item_id,
            price: popupItem.price,
            name: popupItem.name,
            type: popupItem.type,
          }),
        }
      );

      if (response.ok) {
        setData((prevData) => ({
```

```
            ...prevData,
            user: {
              ...prevData.user,
              balance: prevData.user.balance +
popupItem.price,
            },
          }));
          setPopup(false);
          setItems((prevItems) =>
            prevItems.filter((item) => item.item_id !==
popupItem.item_id)
          );
        } else {
          console.error("Помилка при продажі предмета:",
response.statusText);
        }
      } catch (error) {
        console.error("Помилка при продажі предмета:",
error);
      }
    }
  };

  if (status == null || !data) {
    return (
      <div className="spinner-container" style={{
marginTop: "200px" }}>
        <Spinner animation="border" role="status">
          <span className="visually-
hidden">Loading...</span>
        </Spinner>
      </div>
    );
  }

  if (status.code !== 200) {
    return <div>{status.message}</div>;
  }

  const totalPages = Math.ceil(totalItems / itemsPerPage);

  return (
    <>
      <Container className="profile-page mt-4 pb-4">
        {items.length > 0 || searchTerm ? (
          <>
            <Row className="mb-4">
              <Col md={10}>
                <Form.Group controlId="search">
```

```
                  <Form.Control
                    type="text"
                    placeholder="Search..."
                    value={searchTerm}
                    onChange={handleSearchChange}
                  />
                </Form.Group>
              </Col>
              <Col md={2} className="d-flex justify-
content-end">
                <DropdownButton id="dropdown-sort"
title="Сортування">
                  <Dropdown.Item onClick={() =>
handleSortChange("price-asc")}>
                    Ціна за зростанням
                  </Dropdown.Item>
                  <Dropdown.Item onClick={() =>
handleSortChange("price-desc")}>
                    Ціна за спаданням
                  </Dropdown.Item>
                  <Dropdown.Item onClick={() =>
handleSortChange("name-asc")}>
                    Назва за зростанням
                  </Dropdown.Item>
                  <Dropdown.Item onClick={() =>
handleSortChange("name-desc")}>
                    Назва за спаданням
                  </Dropdown.Item>
                </DropdownButton>
              </Col>
            </Row>
            <Row>
              {items.map((item) => (
                <Col key={item.item_id} md={3}
className="mb-4">
                  <Card
                    className="bg-dark text-white item-
card"
                    onClick={() => handleItemClick(item)}
                  >
                    <Card.Img
                      variant="top"
                      src={item.image_url}
                      alt={item.name}
                      className="item-image"
                    />
                    <Card.Body>
                      <Card.Title style={{ color:
`#${item.name_color}` }}>
```

```
                        {item.name}
                      </Card.Title>
                      <Card.Text>{item.type}</Card.Text>
                      {item.price !== 0 ? (
                        <Card.Text>Ціна: {item.price}
грн.</Card.Text>
                      ) : (
                        <Card.Text>Overstock</Card.Text>
                      )}
                    </Card.Body>
                  </Card>
                </Col>
              ))}
            </Row>
            <Pagination className="justify-content-center">
              <Pagination.First
                onClick={() => handlePageChange(1)}
                disabled={currentPage === 1}
              />
              <Pagination.Prev
                onClick={() => handlePageChange(currentPage
- 1)}
                disabled={currentPage === 1}
              />
              {Array.from({ length: totalPages }, (_,
index) => (
                <Pagination.Item
                  key={index + 1}
                  active={index + 1 === currentPage}
                  onClick={() => handlePageChange(index +
1)}
                >
                  {index + 1}
                </Pagination.Item>
              ))}
              <Pagination.Next
                onClick={() => handlePageChange(currentPage
+ 1)}
                disabled={currentPage === totalPages ||
!totalItems}
              />
              <Pagination.Last
                onClick={() =>
handlePageChange(totalPages)}
                disabled={currentPage === totalPages ||
!totalItems}
              />
            </Pagination>
          </>
```

```
        ) : (
          <div style={{marginTop: "185px",
marginLeft:"80px"}}>
            <h1
              style={{
                textAlign: "center",
                fontSize: "28px",
                color: "#4EE0B1",
              }}
            >
              Тут відображатимуться ваші скіни
            </h1>
          </div>
        )}

        <Modal show={popup} onHide={closePopup}>
          {popupItem && (
            <>
              <Modal.Header closeButton>
                <Modal.Title>{popupItem.name}</Modal.Title>
              </Modal.Header>
              <Modal.Body>
                <img
                  src={popupItem.image_url}
                  alt={popupItem.name}
                  className="popup-image mb-3"
                  style={{ width: "100%" }}
                />
                <p style={{ color:
`#${popupItem.name_color}` }}>
                  {popupItem.name}
                </p>
                <p>{popupItem.type}</p>
                {popupItem.price === 0 ? (
                  <p>Неможливо продати</p>
                ) : (
                  <>
                    <p>Ціна: {popupItem.price} грн.</p>
                    <Button
                      variant="primary"
                      key={uuidv4()}
                      onClick={handleSellItem}
                    >
                      Продати
                    </Button>
                  </>
                )}
              </Modal.Body>
            </>
```

```
          )}
        </Modal>
      </Container>
    </>
  );
}
```

- Компонент використовує хуки useEffect та useState для керування станом та виконання запитів до сервера.
- Функція fetchInventory виконує асинхронний запит до сервера для отримання інвентаря користувача, використовуючи fetch з методами GET та відповідними заголовками.
- Результати запиту зберігаються у станах items та totalItems, що дозволяє відображати інвентар користувача.
- Компонент надає можливість пошуку та сортування інвентаря за допомогою полів searchTerm та sortOption.
- Використовується модальне вікно для відображення детальної інформації про обраний предмет з інвентаря.
- Поки виконується запит до сервера, відображається анімація завантаження (Spinner).
- У випадку помилки запиту, відображається сторінка помилки з відповідною інформацією.

Цей компонент відображає аналітичну інформацію для користувача.

Лістинг 3 - код файла AnalyticsPage.jsx:

```
export default function AnalyticsPage() {
  const { data } = useAppContext();
  const [analyticsData, setAnalyticsData] = useState(null);
  const [status, setStatus] = useState(null);

  useEffect(() => {
    async function fetchAnalytics() {
      if (data && data.user && data.user.token &&
data.user.steam_id) {
        try {
          const response = await fetch(
```

```
            `http://127.0.0.1:8000/api/v1/get_transactions_
analytics/`,
            {
              method: "GET",
              headers: {
                token: data.user.token,
                steam: data.user.steam_id,
              },
            }
          );

          if (response.ok) {
            const result = await response.json();

            const formattedData = {
              ...result,
              price_per_date:
result.price_per_date.map(item => ({
                ...item,
                date: moment(item.date).format('YYYY-MM-DD
HH:mm:ss')
              }))
            };

            setAnalyticsData(formattedData);
            setStatus({ code: 200, message: "OK" });
          } else {
            setStatus({ code: response.status, message:
response.statusText });
          }
        } catch (error) {
          setStatus({ code: 503, message: "Service
Unavailable" });
        }
      }
    }

    fetchAnalytics();
  }, [data]);

  if (status == null || !data) {
    return (
      <div className="spinner-container" style={{
marginTop: "200px" }}>
        <Spinner animation="border" role="status">
          <span className="visually-
hidden">Loading...</span>
        </Spinner>
      </div>
```

```
    );
  }

  if (status.code !== 200) {
    return <div>{status.message}</div>;
  }
```

- Компонент використовує хуки useEffect та useState для керування станом та виконання побічних ефектів.
- Під час першого рендеру виконується асинхронний запит до сервера для отримання аналітичних даних.
- Використовується метод fetch з HTTP методом GET та відповідними заголовками, що містять токен користувача і його Steam ID.
- Відповідь сервера обробляється для перевірки статусу: якщо відповідь успішна (статус 200), аналітичні дані зберігаються у стані analytics.
- У випадку помилки, компонент показує відповідну сторінку помилки.
- Компонент також відображає анімацію завантаження, поки очікується відповідь від сервера.
- Якщо запит успішний, відображається аналітична інформація, включаючи загальну кількість предметів та їх загальну вартість.

Частина коду Header.jsx, яка відповідає за верхню частину сторінки сайту:

Лістинг *4 - частина коду файла Header.jsx:*

```
useEffect(() => {
    (async () => {
      let _data = null;
      try {
        _data = await
fetch("http://127.0.0.1:8000/api/v1/get_balance/", {
          method: "GET",
          headers: {
            token: data["user"]["token"],
            steam: data["user"]["steam_id"],
          },
        });
        const response = await _data.json();
        setBalanceStatus({ code: 200, message: "OK" });
        const balanceFromResponse = response["balance"];
```

```
        const roundedBalance =
Math.round(balanceFromResponse * 100) / 100;
        setBalance(roundedBalance);
      } catch (error) {
        if (_data == null)
          setBalanceStatus({ code: 503, message: "Service
Unavailable" });
        else if (_data.status == 401) {
          setBalanceStatus({ code: 403, message:
"Forbidden" });
        }
      }
    })();
  }, [data]);

  const handleWithdraw = () => {
    setShowModal(true);
  };

  const handlePaymentSuccess = () => {
    setBalance(Math.round((balance - withdrawAmount) * 100)
/ 100);
    setShowModal(false);
    console.log("Payment successful");
  };

  if (balanceStatus == null) return "Loading...";

  return (
    <>
      <Navbar className="transparent-navbar" expand="lg">
        <Container>
          <Navbar.Brand href="/" className="d-flex align-
items-center">
            <Icon
              icon="akar-icons:home"
              style={{ fontSize: "2rem", color: "#74CBAF"
}}
            />
            <span
              className="ms-2"
              style={{ fontSize: "1.5rem", color: "#74CBAF"
}}
            >
              Домашня сторінка
            </span>
          </Navbar.Brand>
          <Navbar.Toggle aria-controls="basic-navbar-nav"
/>
```

```
          <Navbar.Collapse id="basic-navbar-nav">
            <Nav className="ms-auto">
              {isAuthenticated && (
                <>
                  <Nav.Item
                    className="navbar-text"
                    style={{ marginTop: "19px" }}
                  >
                    Баланс: {balance} грн.
                  </Nav.Item>
                  <Nav.Link>
                    <Button variant="primary"
onClick={handleWithdraw}>
                      Зняти кошти
                    </Button>
                  </Nav.Link>
                </>
              )}
              <Nav.Link href="https://t.me/review_lxnmnk"
target="_blank">
                <Button variant="primary">Відгуки</Button>
              </Nav.Link>
              {isAuthenticated ? (
                <>
                  <Nav.Link href="/profile">
                    <Button
variant="primary">Продаж</Button>
                  </Nav.Link>
                  <Nav.Link href="/transactions">
                    <Button variant="primary">Історія
транзакцій</Button>
                  </Nav.Link>
                  <Nav.Link href="/analytics">
                    <Button
variant="primary">Аналітика</Button>
                  </Nav.Link>

                  <Nav.Link
                    onClick={() => {
                      setData({});
                      navigate("/");
                    }}
                  >
                    <Button variant="danger">Вихід</Button>
                  </Nav.Link>
                </>
              ) : (
                <Nav.Link
href="https://steamcommunity.com/openid/login?openid.ns=htt
```

```
p://specs.openid.net/auth/2.0&openid.identity=http://specs.
openid.net/auth/2.0/identifier_select&openid.claimed_id=htt
p://specs.openid.net/auth/2.0/identifier_select&openid.mode
=checkid_setup&openid.return_to=http://localhost:5173/login
&openid.realm=http://localhost:5173">
                  <Button variant="primary">Увійти зі
Steam</Button>
                </Nav.Link>
              )}
            </Nav>
          </Navbar.Collapse>
        </Container>
      </Navbar>

      <Modal show={showModal} onHide={() =>
setShowModal(false)}>
        <Modal.Header closeButton>
          <Modal.Title>Зняття коштів</Modal.Title>
        </Modal.Header>
        <Modal.Body>
          <Form.Group controlId="formWithdrawAmount">
            <Form.Label>Сума зняття</Form.Label>
            <Form.Control
              type="number"
              placeholder="Введіть суму"
              value={withdrawAmount}
              onChange={(e) =>
setWithdrawAmount(e.target.value)}
            />
          </Form.Group>
          <StripeCheckoutButton
            currentAmount={balance}
            amount={withdrawAmount}
            onSuccess={handlePaymentSuccess}
          />
        </Modal.Body>
      </Modal>
    </>
  );
```

- Компонент використовує хуки useState для керування станом модального вікна та суми зняття.
- Імпортуються необхідні компоненти з react-bootstrap для створення навігаційної панелі та модального вікна.
- Використовується useNavigate з react-router-dom для навігації

між сторінками.

- В компоненті є навігаційна панель (Navbar), яка містить посилання на різні сторінки додатку, такі як профіль, продаж, транзакції та аналітика.
- Якщо користувач залогінений, відображаються кнопки для зняття коштів та виходу з аккаунта. В іншому випадку відображається кнопка для логіну через Steam.
- Модальне вікно (Modal) використовується для введення суми зняття та підтвердження операції зняття коштів.
- Використовується компонент StripeCheckoutButton для інтеграції з платіжною системою Stripe та обробки успішних платежів

Цей компонент відповідає за відображення історії транзакцій користувача.

Лістинг 5 - *частина коду файла Transactions.jsx:*

```
export default function TransactionsPage() {
  const { data } = useAppContext();
  const [transactions, setTransactions] = useState([]);
  const [status, setStatus] = useState(null);

  useEffect(() => {
    async function fetchTransactions() {
      if (data && data.user && data.user.token &&
data.user.steam_id) {
        try {
          const response = await fetch(
            `http://127.0.0.1:8000/api/v1/get_transactions/
`,
            {
              method: "GET",
              headers: {
                token: data.user.token,
                steam: data.user.steam_id,
              },
            }
          );

          if (response.ok) {
            const result = await response.json();
            setTransactions(result);
            setStatus({ code: 200, message: "OK" });
          } else {
            setStatus({ code: response.status, message:
response.statusText });
          }
        } catch (error) {
          setStatus({ code: 503, message: "Service
Unavailable" });
        }
      }
    }

    fetchTransactions();
  }, [data]);

  if (status == null || !data) {
    return (
```

```
      <div className="spinner-container" style={{
marginTop: "200px" }}>
        <Spinner animation="border" role="status">
          <span className="visually-
hidden">Loading...</span>
        </Spinner>
      </div>
    );
  }

  if (status.code !== 200) {
    return <div>{status.message}</div>;
  }

  return (
    <Container className="transactions-page mt-4 pb-4">
      <h1
        style={{
          textAlign: "center",
          fontSize: "28px",
          color: "#4EE0B1",
          marginBottom: "20px",
        }}
      >
        Ваші транзакції
      </h1>
      <Table striped bordered hover>
        <thead>
          <tr>
            <th>Item ID</th>
            <th>Ціна</th>
            <th>Назва</th>
            <th>Тип</th>
            <th>Дата продажу</th>
          </tr>
        </thead>
        <tbody>
          {transactions.map((transaction) => (
            <tr key={transaction.item_id}>
              <td>{transaction.item_id}</td>
              <td>{transaction.price} грн.</td>
              <td>{transaction.name}</td>
              <td>{transaction.type}</td>
              <td>{new
Date(transaction.date_sold).toLocaleString()}</td>
            </tr>
          ))}
        </tbody>
      </Table>
```

```
    </Container>
  );
}
```

- Компонент використовує хуки useEffect та useState для керування станом та виконання побічних ефектів.
- Під час першого рендеру виконується асинхронний запит до сервера для отримання історії транзакцій.
- Використовується метод fetch з HTTP методом GET та відповідними заголовками, що містять токен користувача і його Steam ID.
- Відповідь сервера обробляється для перевірки статусу: якщо відповідь успішна (статус 200), історія транзакцій зберігається у стані transactions.
- У випадку помилки, компонент показує відповідну сторінку помилки.
- Компонент також відображає анімацію завантаження, поки очікується відповідь від сервера.
- Якщо запит успішний, відображається таблиця з історією транзакцій, включаючи ID, дату, суму та статус транзакцій.

3.2 Розробка бек-енду

Бек-енд веб-сервісу буде розроблено за допомогою фреймворку Django. Django - це Python-фреймворк для розробки веб-додатків. Він пропонує широкий спектр функцій, таких як система аутентифікації, система авторизації, система адміністрування та система обробки платежів.

Основні компоненти бек-енду:

- Моделі: Моделі будуть використовуватися для представлення даних про ігрові артефакти, користувачів та замовлення.
- Серіалізатори: Серіалізатори будуть використовуватися для перетворення моделей у JSON-формат для передачі даних між фронт-ендом та бек-ендом.
- Представлення: Представлення будуть використовуватися для обробки HTTP-запитів та відповідей.
- Сигнальні системи: Сигнальні системи будуть використовуватися для оновлення інтерфейсу користувача при зміні даних.

3.2.1 Розгляд основних частин бек-енду

Фрагмент коду manage.py, який дозволяє виконувати завдання, такі як створення бази даних, міграції, запуск сервера розробки тощо, з використанням командного рядка:

Лістинг *6 - частина коду файла manage.py:*

```
"""Django's command-line utility for administrative
tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'skin_trade.settings')
    try:
```

```
        from django.core.management import
execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's
installed and "
            "available on your PYTHONPATH environment
variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Цей скрипт, командний інтерфейс Django, який дозволяє виконувати різні адміністративні завдання, такі як створення бази даних, міграції, запуск сервера розробки тощо, з використанням командного рядка. Він встановлює змінну середовища для налаштування Django та запускає виконання вказаних завдань, переданих через аргументи командного рядка.

Цей клас обробляє аутентифікацію користувачів через Steam і повертає JWT токен.

Лістинг *7 – Клас SteamAuthAPIView:*

```
class SteamAuthAPIView(APIView):

    def get(self, request):
        serializer = SteamAuthSerializer(data=request.GET)
        serializer.is_valid(raise_exception=True)

        steam_id = request.GET['identity'].split('/')[-1]
        user, created =
SteamUser.objects.get_or_create(steam_id=steam_id)

        if created:
            user.balance = 0
            user.save()

        token = user.generate_jwt_token()
        cache.set(steam_id, str(token), timeout=3600)
```

```
        steam_api_key = settings.STEAM_API_KEY
        user_data = self.get_steam_user_data(steam_id,
steam_api_key)

        return Response({
            'token': token,
            'steam_id': steam_id,
            'steam_user_data': user_data,
            'balance': user.balance
        }, status=status.HTTP_200_OK)

    def get_steam_user_data(self, steam_id, api_key):
        url =
f"http://api.steampowered.com/ISteamUser/GetPlayerSummaries
/v0002/?key={api_key}&steamids={steam_id}"
        response = requests.get(url)
        if response.status_code == 200:
            data = response.json()
            if 'response' in data and 'players' in
data['response'] and len(data['response']['players']) > 0:
                return data['response']['players'][0]
            else:
                return {}
        else:
            return None
```

Цей клас API виконує аутентифікацію користувачів через Steam, створює нового користувача, якщо він не існує, та повертає JWT токен. Метод get отримує дані користувача зі Steam API, створює або отримує користувача з бази даних, генерує токен і кешує його.

Цей клас обробляє вихід користувача, видаляючи його токен з кешу.

Лістинг 8 - *Клас LogoutAPIView:*

```
class LogoutAPIView(APIView):

    @method_decorator(custom_login_required)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def get(self, request):
        cache.delete(request.headers.get('steam'))
```

```
        return Response('Logout successful.',
status=status.HTTP_200_OK)
```

Вихід із системи: Клас LogoutAPIView відповідає за видалення токену доступу з кешування, коли користувач виходить із системи.

Фрагмент коду з файлу views.py, який відповідає за відображення баланса користувача на сайті:

Лістинг *9* - *Клас GetBalanceAPIView:*

```
class GetBalanceAPIView(APIView):

    @method_decorator(custom_login_required)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def get(self, request):
        steam_id = request.headers.get('steam')
        try:
            user = SteamUser.objects.get(steam_id=steam_id)
            return Response({'balance': user.balance},
status=status.HTTP_200_OK)
        except SteamUser.DoesNotExist:
            return Response({'error': 'User not found'},
status=status.HTTP_404_NOT_FOUND)
                        status=status.HTTP_200_OK)
```

Фрагмент коду з файлу views.py, який відповідає за продаж ігрових артефактів на сайту:

Лістинг *10* - *Клас SellItemAPIView:*

```
class SellItemAPIView(APIView):

    @method_decorator(custom_login_required)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def post(self, request):
        try:
            steam = request.headers.get('steam')
            token = request.headers.get('token')
            item_id = request.data.get('item_id')
            price = request.data.get('price')
```

```
            name = request.data.get('name')
            type = request.data.get('type')

            logger.debug(f'Received steam ID: {steam}')
            logger.debug(f'Received steam TOKEN: {token}')
            logger.debug(f'Received item ID: {item_id}')
            logger.debug(f'Received price: {price}')
            logger.debug(f'Received name: {name}')
            logger.debug(f'Received type: {type}')

            if not steam or not token or not item_id or not
price:
                return Response({'error': 'Missing required
parameters'}, status=status.HTTP_400_BAD_REQUEST)

            user = SteamUser.objects.get(steam_id=steam)
            item =
Inventory.objects.filter(steam_user=user,
item_id=item_id).first()

            if not item:
                return Response({'error': 'Item not
found'}, status=status.HTTP_404_NOT_FOUND)

            transaction = Transaction(
                seller=user,
                item_id=item_id,
                price=price,
                name=name,
                type=type,
            )
            transaction.save()

            user.balance += Decimal(price)
            user.save()

            item.delete()

            cache.delete_pattern(f'inventory_{steam}_*')

            return Response({'message': 'Item sold
successfully'}, status=status.HTTP_200_OK)

        except Exception as e:
            logger.error(f'An error occurred: {e}',
exc_info=True)
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Цей клас API обробляє продаж предметів з інвентаря користувача. Метод ‘post’ перевіряє наявність необхідних параметрів, шукає користувача та предмет в базі даних, створює транзакцію, оновлює баланс користувача, видаляє проданий предмет з інвентаря та очищає кеш. У випадку помилки повертає відповідне повідомлення про помилку.

Фрагмент коду з файлу views.py, який відповідає за відображення предметів на сайті:

Лістинг *11 - Клас GetInventoryAPIView:*

```
class GetInventoryAPIView(APIView):

    @method_decorator(custom_login_required)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)

    def get(self, request):
        try:
            steam = request.headers.get('steam')
            token = request.headers.get('token')
            search = request.query_params.get('search', '')
            page = int(request.query_params.get('page', 1))
            limit = int(request.query_params.get('limit',
10))
            sort_by = request.query_params.get('sort_by',
'name')
            order = request.query_params.get('order',
'asc')
            offset = (page - 1) * limit

            if not steam or not token:
                return Response({'error': 'Steam ID or
token missing'}, status=status.HTTP_400_BAD_REQUEST)

            cache_key = f'inventory_{steam}'
            cached_response = cache.get(cache_key)

            user = SteamUser.objects.get(steam_id=steam)
            totalTransactionsCount =
Transaction.objects.filter(seller=user).count()
            totalInventoryCount =
Inventory.objects.filter(steam_user=user).count()

            if cached_response:
```

```
                totalCachedCount =
len(cached_response.get('descriptions', []))
            else:
                totalCachedCount = 0

            if totalCachedCount == totalTransactionsCount
and totalCachedCount > 0:
                return Response({'descriptions': [],
'total_items': 0, 'page': page, 'limit': limit},
status=status.HTTP_200_OK)

            if cached_response and totalInventoryCount == 0
and totalTransactionsCount == 0:
                descriptions =
cached_response.get('descriptions', [])

                inventory_objects = []
                for item in descriptions:
                    inventory_objects.append(Inventory(
                        steam_user=user,
                        item_id=item.get('item_id'),
                        market_hash_name=item.get('market_h
ash_name'),
                        tradable=item.get('tradable'),
                        price=item.get('price'),
                        name=item.get('name'),
                        description=item.get('description')
,
                        image_url=item.get('image_url'),
                        type=item.get('type'),
                    ))

                Inventory.objects.bulk_create(inventory_obj
ects)

                return Response(cached_response,
status=status.HTTP_200_OK)

            if cached_response:
                print('Cached response found')
                inventory_items =
Inventory.objects.filter(steam_user=user)
                if search:
                    inventory_items =
inventory_items.filter(name__icontains=search)

                if sort_by == 'price':
```

```
                    inventory_items =
inventory_items.order_by('-price' if order == 'desc' else
'price')
                elif sort_by == 'name':
                    inventory_items =
inventory_items.order_by('-name' if order == 'desc' else
'name')

                total_items = inventory_items.count()
                paginated_items =
inventory_items[offset:offset + limit]

                response_data = {
                    'descriptions':
list(paginated_items.values('item_id', 'market_hash_name',
'tradable', 'price', 'name', 'description', 'image_url',
'type')),
                    'total_items': total_items,
                    'page': page,
                    'limit': limit
                }

                return Response(response_data,
status=status.HTTP_200_OK)

            user_items_response =
requests.get(f'https://steamcommunity.com/inventory/{steam}
/730/2?l=ukrainian&count=300')
            if user_items_response.status_code == 401:
                logger.error('Unauthorized access to Steam
API. Ensure that the user inventory is public.')
                return Response({'error': 'Unauthorized
access to Steam API. Ensure that the user inventory is
public.'}, status=status.HTTP_401_UNAUTHORIZED)
            user_items_response.raise_for_status()

            try:
                user_items = user_items_response.json()
                logger.debug(f'User items response:
{user_items}')
            except json.JSONDecodeError as e:
                logger.error(f'Error decoding JSON response
for user items: {e}')
                return Response({'error': 'Invalid response
from Steam API'}, status=status.HTTP_502_BAD_GATEWAY)

            tradable_items = [item for item in
user_items['descriptions'] if item.get('tradable') == 1]
            prices = get_item_prices(tradable_items)
```

```
            Inventory.objects.filter(steam_user=user).delet
e()

            inventory_objects = []
            for item in tradable_items:
                market_hash_name =
item.get('market_hash_name')
                price = prices.get(market_hash_name)
                if price:
                    price = float(price.replace('₴',
'').replace('$', '').replace(',', '.').strip())
                image_url = f"https://steamcommunity-
a.akamaihd.net/economy/image/{item.get('icon_url')}"
                inventory_objects.append(Inventory(
                    steam_user=user,
                    item_id=item.get('classid'),
                    market_hash_name=market_hash_name,
                    tradable=True,
                    price=(price * 40.2 if price is not
None else 0),
                    name=item.get('name'),
                    description=item.get('descriptions'),
                    image_url=image_url,
                    type=item.get('type'),
                ))

            Inventory.objects.bulk_create(inventory_objects
)

            inventory_items =
Inventory.objects.filter(steam_user=user)
            if search:
                inventory_items =
inventory_items.filter(name__icontains=search)

            if sort_by == 'price':
                inventory_items =
inventory_items.order_by('-price' if order == 'desc' else
'price')
            elif sort_by == 'name':
                inventory_items =
inventory_items.order_by('-name' if order == 'desc' else
'name')

            total_items = inventory_items.count()
            paginated_items = inventory_items[offset:offset
+ limit]
```

```
            response_data = {
                'descriptions':
list(paginated_items.values('item_id', 'market_hash_name',
'tradable', 'price', 'name', 'description', 'image_url',
'type')),
                'total_items': total_items,
                'page': page,
                'limit': limit
            }

            cache.set(cache_key, response_data,
timeout=60*60*10)

            return Response(response_data,
status=status.HTTP_200_OK)

        except requests.HTTPError as e:
            logger.error(f'HTTP request failed: {e}',
exc_info=True)
            if e.response.status_code == 401:
                return Response({'error': 'Unauthorized
access to Steam API. Ensure that the user inventory is
public.'}, status=status.HTTP_401_UNAUTHORIZED)
            else:
                return Response({'error': 'Failed to fetch
data from external APIs'},
status=status.HTTP_502_BAD_GATEWAY)

        except Exception as e:
            logger.error(f'An error occurred: {e}',
exc_info=True)
            return Response({'error': str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Цей клас API повертає інвентар користувача з можливістю пошуку, сортування та пагінації. Метод `get` отримує параметри запиту, шукає користувача в базі даних та інвентар, застосовує фільтри пошуку та сортування, обчислює загальну кількість предметів і повертає відповідні результати. У випадку помилки повертає відповідне повідомлення про помилку.

Кожен клас API використовується для виконання конкретного завдання та взаємодії з іншими частинами системи, такими як база даних, зовнішні API та кешування. Це забезпечує функціональність, необхідну для роботи веб-

додатку з платформою Steam.

Фрагмент коду на мові програмування Python. Клас визначає серіалізатор для аутентифікації через Steam.

Лістинг *12 – Код файлу serializers.py:*

```
import requests
from rest_framework import serializers, status


class SteamAuthSerializer(serializers.Serializer):
    claimed_id = serializers.CharField(max_length=255,
write_only=True)
    identity = serializers.CharField(max_length=255,
write_only=True)
    return_to = serializers.CharField(max_length=255,
write_only=True)
    response_nonce = serializers.CharField(max_length=255,
write_only=True)
    assoc_handle = serializers.CharField(max_length=255,
write_only=True)
    signed = serializers.CharField(max_length=255,
write_only=True)
    sig = serializers.CharField(max_length=255,
write_only=True)

    def validate(self, data):
        check_data = {'openid.ns':
'http://specs.openid.net/auth/2.0',
                      'openid.mode':
'check_authentication',
                      'openid.op_endpoint':
'https://steamcommunity.com/openid/login',

                      'openid.claimed_id':
data.get('claimed_id'),
                      'openid.identity':
data.get('identity'),
                      'openid.return_to':
data.get('return_to'),
                      'openid.response_nonce':
data.get('response_nonce').replace(" ", "+"),
                      'openid.assoc_handle':
data.get('assoc_handle'),
                      'openid.signed': data.get('signed'),
```

```
                        'openid.sig':
data.get('sig').replace(" ", "+"),
                        }

        result = requests.get(
            url='https://steamcommunity.com/openid/login',
            params=check_data
        ).text.split('\n')[1]

        if result != 'is_valid:true':
            raise serializers.ValidationError("Bad OpenId
params.", status.HTTP_400_BAD_REQUEST)

        return {}
```

Цей код містить серіалайзер SteamAuthSerializer, який використовується для перевірки параметрів автентифікації користувачів через Steam OpenID. Серіалайзер приймає різні параметри, такі як ідентифікатори, nonce та підписи, і виконує запит до сервера автентифікації Steam, щоб перевірити їх правильність. Якщо перевірка успішна, серіалайзер повертає порожній словник. У випадку невідповідності отриманих параметрів вимогам, викидається виняток ValidationError з відповідним повідомленням про помилку. Цей серіалайзер забезпечує коректну обробку та перевірку даних, що надходять з боку користувача під час процесу автентифікації.

3.3 Тестування веб-сервісу

На рисунку 3.13 можна побачити головне вікно запущеного веб-сервісу продажу ігрових артефактів з використанням фреймворків React та Django.

Рисунок *3.13 — Головне вікно веб-сервісу*

На рисунку 3.14 зображено інтерфейс веб-сервісу після входу користувача до системи, що відображає доступні розділи.

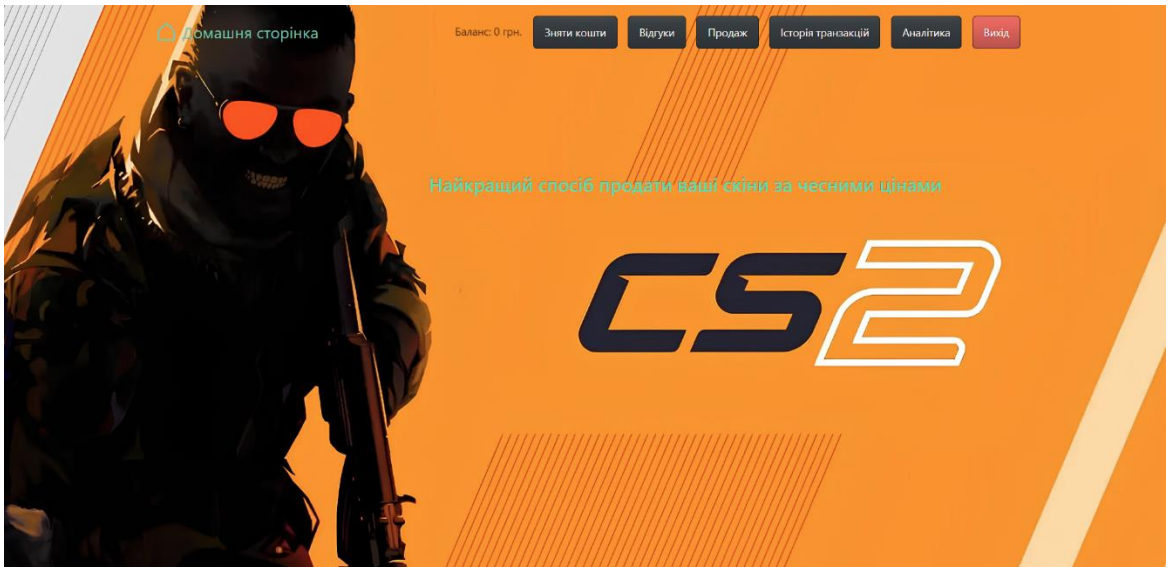

Рисунок *3.14 — Інтерфейс після входу користувача*

На рисунку 3.15 представлено вікно інвентаря користувача, де відображаються всі його предмети з можливістю сортування та пошуку.

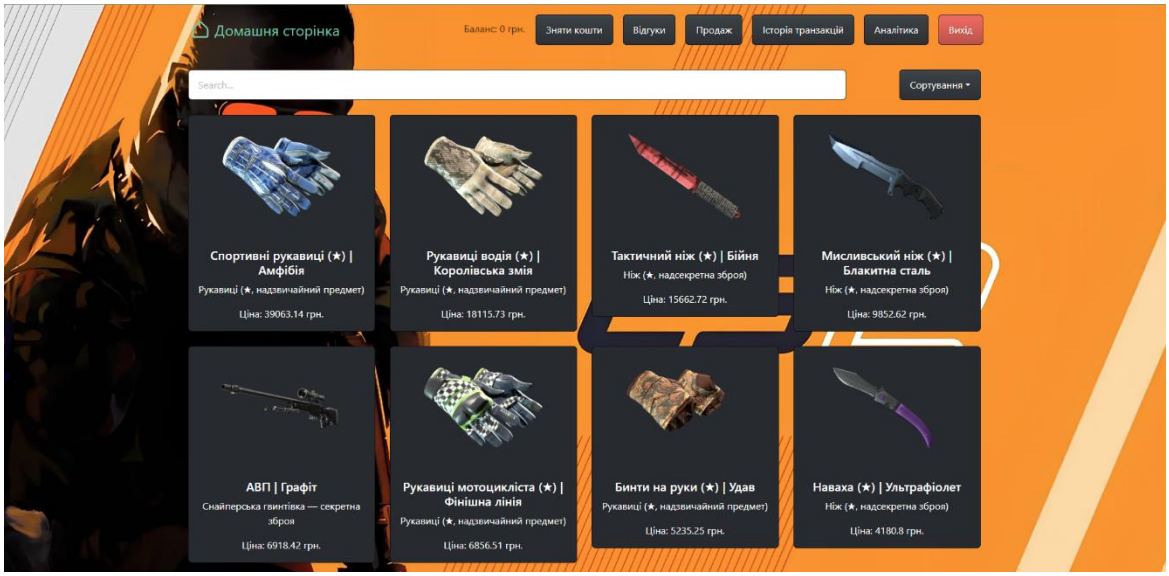

Рисунок *3.15 — Вікно інвентаря користувача*

На рисунку 3.16 можна побачити результат пошуку конкретного предмета в інвентарі користувача.

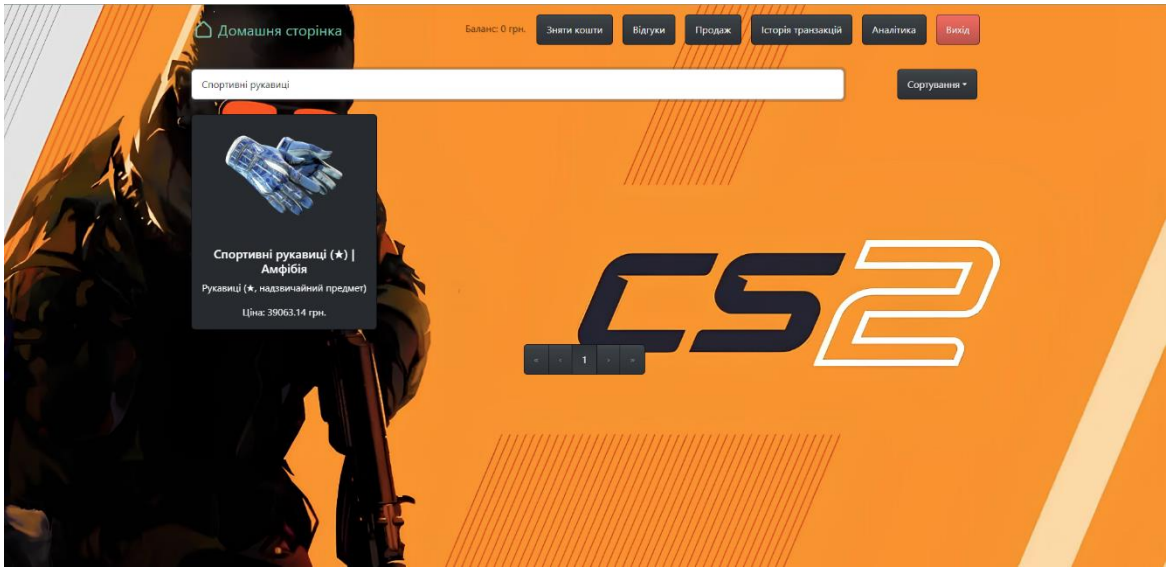

Рисунок *3.16 — Результат пошуку предмета в інвентарі*

На рисунку 3.17 зображено історію транзакцій користувача, яка включає інформацію про всі його продажі.

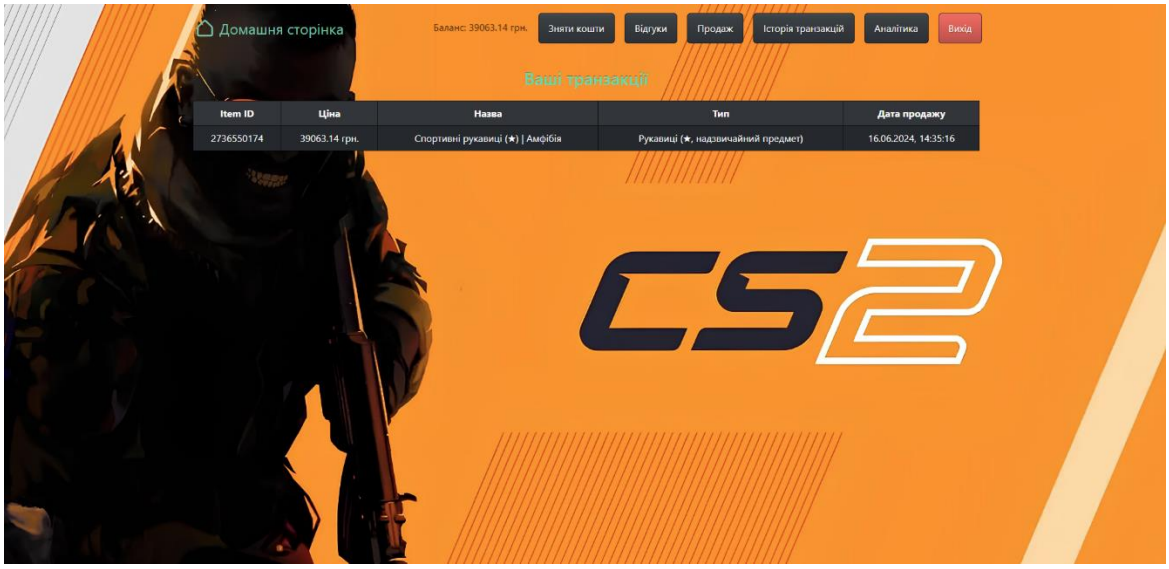

Рисунок *3.17 — Історія транзакцій користувача*

На рисунку 3.18 представлено вікно аналітики транзакцій користувача, яке відображає графік продажів по датах та типах.

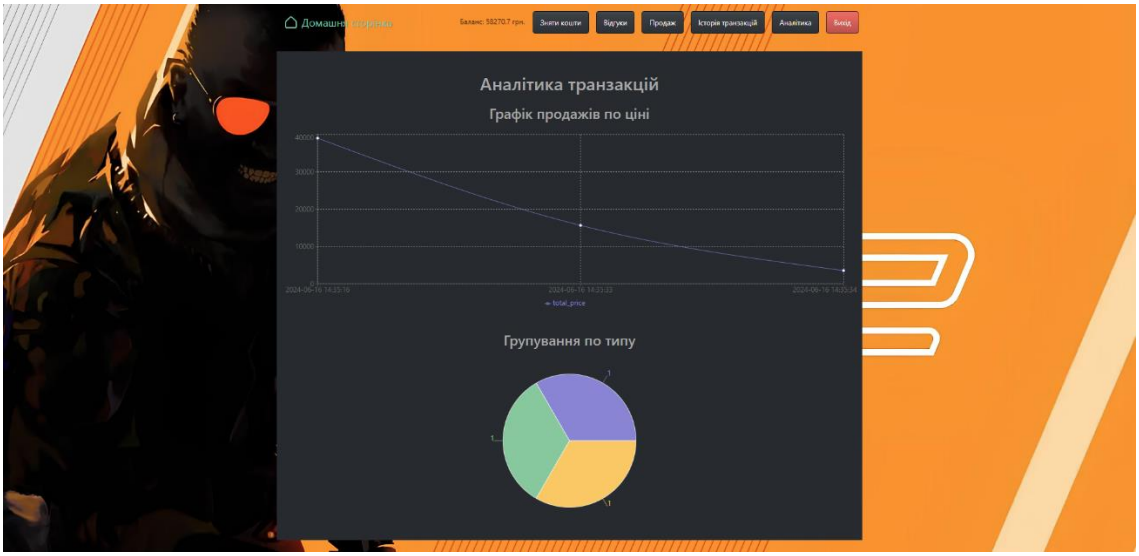

Рисунок 3.18 — *Вікно аналітики транзакцій*

На рисунку 3.19 можна побачити модальне вікно зняття коштів з балансу користувача, яке дозволяє ввести суму для зняття та підтвердити операцію.

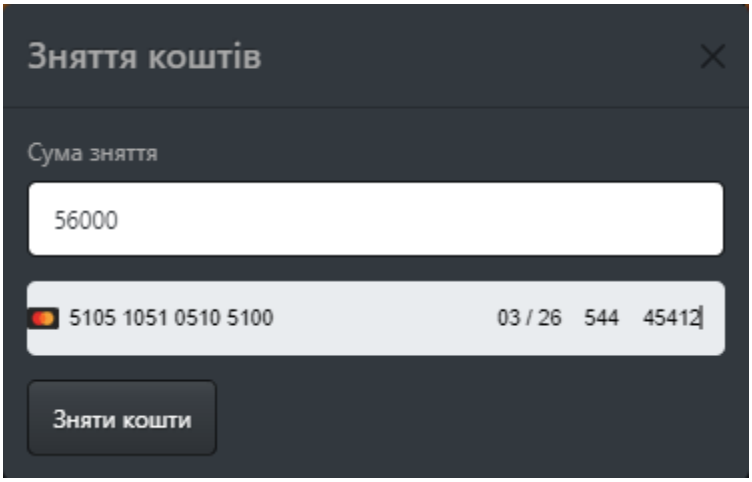

Рисунок *3.19 — Модальне вікно зняття коштів*

Загалом тестування пройшло успішно, і всі основні аспекти функціональності додатку були вдало перевірені. Усі компоненти React працюють правильно, і взаємодіють з користувачем як очікується. Комунікація з сервером також відбувається без помилок, і дані правильно відображаються на клієнтському інтерфейсі. Навігація працює коректно, і користувач може легко переміщатися між різними сторінками додатку. Також було успішно перевірено стабільність додатку на різних пристроях та браузерах. Загалом, додаток готовий до використання, і відповідає всім вимогам функціональності і безпеки.

3.4 Висновок за розділом 3

У цьому розділі було описано розробку та тестування веб-сервісу для продажу ігрових артефактів. React забезпечує оптимізований інтерфейс користувача для фронт-енду, тоді як Django з його багатим набором функцій відповідає за бек-енд. Ретельне тестування на трьох рівнях гарантує

стабільність та коректність роботи сервісу. Розгортання на AWS Elastic Beanstalk забезпечить масштабування та доступність.

В результаті створено готовий до використання веб-сервіс, що відповідає усім вимогам функціональності та безпеки. Він може стати цінним інструментом для геймерів, пропонуючи зручний інтерфейс, широкий спектр функцій та високий рівень безпеки.

ВИСНОВКИ

1. У ході виконання кваліфікаційної роботи було детально досліджено концепцію веб-сервісу та його особливості, розглянуті аналоги, а також архітектуру та технології розробки веб-сервісів, включаючи фреймворки React і Django. Процес розробки веб-сервісу було розглянуто від вибору технологій та проектування бази даних до розробки фронтенду та бекенду, їх інтеграції, тестування і налагодження.
2. У розділі 3 було представлено процес розробки фронтенду та бекенду, описано їхні основні частини, а також проведено тестування веб-сервісу. Цей етап роботи дозволив поглибити розуміння процесу розробки програмного забезпечення та використати отримані знання для створення функціонального та надійного веб-сервісу.
3. В результаті виконання кваліфікаційної роботи було успішно реалізовано веб-сервіс з використанням сучасних технологій, що відповідає вимогам сучасного ринку програмного забезпечення. Розроблений веб-сервіс виявився функціональним, ефективним та готовим до використання в реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React: З чого розпочати: Створення веб-застосунків з React та Redux автора Робіна Віруха (2016) (дата звернення: 05.04.2024)

2. Django для початківців: Створення потужних веб-застосунків з Python автора Вільяма С. Вінсента (2018) (дата звернення: 08.04.2024)

3. Проектування RESTful API: Як проектувати, створювати та документувати REST API авторів Леонарда Річардсона та Девіда Рубі (2013) (дата звернення: 20.04.2024)

4. Створення масштабованих веб-застосунків з Django та React автора Дейва Берка (2018) (дата звернення: 21.04.2024)

5. Створення REST API з Django та React автора Крістофера Чудзіцького. URL: https://medium.com/@jadhavmanoj/get-started-with-django-react-js-b8e9280216dd (дата звернення: 20.04.2024)

6. Створення веб-застосунку з React та Django автора Андрія Шумієва. URL: https://ramkumar-m.medium.com/build-web-application-using-react-and-django-69cb37bb3166 (дата звернення: 17.04.2024)

7. Науменко О.В., Скрипник І.А., доцент, к.ф.-м.н.. Веб-сервіс для продажу ігрових артефактів з використанням фреймворків react та django. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізький національний університет. Запоріжжя: ЗНУ, 2024. Т.5. С. 190-191.

8. Real World Django: Створення веб-застосунків для масштабних проектів автора Мелані МакГрат (2020) (дата звернення: 15.04.2024)

9. Advanced React and Redux: Розширені техніки для побудови динамічних веб-застосунків автора Стефена Грідера (2019) (дата звернення: 19.04.2024)

10. Two Scoops of Django 3.x: Кращі практики для розробки Django проектів авторів Деніела Фельдвіля та Одрі Рой Грінфілд (2020) (дата звернення: 22.04.2024)

11. React and Django: Повний посібник по створенню сучасних веб-додатків автора Михайла Іванова (2021) (дата звернення: 25.04.2024)

12. Джанго і React: Практичне керівництво по створенню динамічних веб-сайтів автора Джейсона Брауна. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Django/React (дата звернення: 27.04.2024)

13. Full-Stack React Projects: Створення сучасних веб-застосунків з використанням React та Django автора Саймона Холдера (2021) (дата звернення: 29.04.2024)

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Науменко Олександр Володимирович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-118@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Веб-сервіс для продажу ігрових артефактів з використанням фреймворків React та Django**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 17.06.2024 Підпис _____ Науменко Олександр Володимирович
(студент)

Дата 17.06.2024 Підпис _____ Скрипник Ірина Анатоліївна
(науковий керівник)