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 **Розробка веб-сайту для пошуку та аналізу ігор за широким спектром характеристик**

Виконав: студент 4 курсу, групи 6.1211-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
В.В. Тішков
(ініціали та прізвище)
Керівник к.т.н., доцент О.М. Міхайлуца
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий

Запоріжжя

2025

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 (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Тетяна КРИТСЬКА

“ 01 ” березня 2025 року

З А В Д А Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Тішкову Владиславу Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка веб-сайту для пошуку та аналізу ігор за широким спектром характеристик

   Керівник роботи Міхайлуца Олена Миколаївна, к.т.н., доцент
   (прізвище, ім’я, по батькові, наукова ступінь,вчене звання)
   Затверджені наказом ЗНУ від 27.12.2024 р. № 2029-с

2. Строк подання студентом кваліфікаційної роботи 02.06.2025 р.
3. Вихідні дані бакалаврської 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 збору та аналізу даних з веб-сторінок;
   - створення програмного продукту та його опис;
   - дослідження поставленої проблеми та розробка висновків
5. Перелік графічного матеріалу(з точним зазначення обов’язкових креслень)_____

6. Консультанти розділів бакалаврської роботи

Розділ	Призвіще, ініціали та посада консультанта	Підпис, дата Задвання прийня

7. Дата видачі завдання 01.03.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	01.03.2025	виконано
2	Формулювання мети та основних завдань проєкту	20.03.2025	виконано
3	Огляд існуючих рішень і сервісів для порівняння відеоігор	02.04.2025	виконано
4	Проєктування архітектури бази даних (моделі Django)	07.04.2025	виконано
5	Розробка модуля імпорту даних із RAWG API	15.04.2025	виконано
6	Реалізація шаблону списку ігор з фільтрацією та пошуком	21.04.2025	виконано
7	Реалізація детальної сторінки гри (опис, трейлери, скриншоти)	03.05.2025	виконано
8	Впровадження системи реєстрації, авторизації та профілю	05.05.2025	виконано
9	Реалізація функціоналу «Улюблені ігри» та рейтингу	10.05.2025	виконано
10	Налаштування локалізації (українська/англійська)	28.04.2025	виконано
11	Тестування базових функцій і виправлення помилок	25.05.2025	виконано
12	Написання та оформлення розділів дипломної роботи	10.05.2025	виконано
13	Підготовка презентації, отримання рецензій та захист	01.06.2025	виконано

Студент _____ Тішков В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Міхайлуца О.М.
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Сторінок — 63

Рисунків — 11

Джерел — 15

Тішков В.В Розробка веб-застосунку для пошуку та аналізу ігор за широким спектром характеристик: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / науковий керівник О. М. Міхайлуца Запоріжжя : ЗНУ 2025. 63 с.

Мета роботи полягає у створенні сучасного інструменту для швидкого отримання й порівняння даних про відеоігри різних платформ. Відпрацьовано інтеграцію з RAWG API та Steam, реалізовано бекенд на Django з моделями Game, Platform, Genre, Price, Rating, Screenshot і FavoriteGame у реляційній базі PostgreSQL.

Розроблено адаптивний фронтенд на Bootstrap 5 і JavaScript (ES6, AJAX), що забезпечує динамічне фільтрування за жанрами, платформами та цінами, відкладене завантаження зображень і карусель медіаконтенту з трейлерами та скриншотами. Для прискорення відповіді застосовано кешування у Redis і оптимізовані ORM-запити з використанням Subquery.

Забезпечено безпеку обміну даними: CSRF-захист, екранування шаблонізатором, параметризовані ORM-запити, SSL/HSTS. Функціонал «Улюблені ігри» і рейтингів реалізовано через AJAX-view-функції з методами.

Ключові слова: *веб-застосунок, Django, RAWG API, відеоігри, фільтрація, кешування, AJAX, DDocker Compose, Bootstrap 5, PostgreSQL.*

ABSTRACT

Pages — 63

Figures — 11

Source — 15

Tishkov V.V. Development of a web application for searching and analyzing games by a wide range of characteristics: bachelor's thesis, specialty 121 "Software Engineering" / scientific supervisor O. M. Mikhaylutsa Zaporizhzhia: ZNU 2025. 63 p.

The purpose of the work is to create a modern tool for quickly obtaining and comparing data on video games of different platforms. Integration with RAWG API and Steam has been worked out, a backend on Django with Game, Platform, Genre, Price, Rating, Screenshot and FavoriteGame models in the PostgreSQL relational database has been implemented.

An adaptive frontend on Bootstrap 5 and JavaScript (ES6, AJAX) has been developed, which provides dynamic filtering by genres, platforms and prices, deferred image loading and a media content carousel with trailers and screenshots. To speed up the response, caching in Redis and optimized ORM queries using Subquery have been used.

Data exchange security is ensured: CSRF protection, template shielding, parameterized ORM queries, SSL/HSTS. The "Favorite Games" and ratings functionality is implemented via AJAX view functions with methods.

Keywords: web application, Django, RAWG API, video games, filtering, caching, AJAX, DDocker Compose, Bootstrap 5, PostgreSQL.

ЗМІСТ

ВСТУП 8
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 13
1.1 Огляд літературних джерел 13
1.2 Аналіз програмного забезпечення 13
1.2.1 IsThereAnyDeal 14
1.2.2 CheapShark 15
1.2.3 PriceCharting 15
1.3 Постановка завдання 17
1.4 Висновки до розділу 1 18
2 ВИМОГИ ДО ОТОЧЕННЯ 19
2.1 Аналіз сучасних технологій для розробки веб-застосунку 19
2.2 Аналіз вимог до користувачів веб-застосунку 20
2.3 Аналіз вимог до апаратного та програмного забезпечення 21
2.4 Висновки до розділу 2 22
3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ 23
3.1 Опис предметної області 23
3.2 Архітектура системи 24
3.2.1 Компонентна діаграма 24
3.2.2 Діаграма розгортання 27
3.2.3 Опис Use-case діаграми 27
3.3 Схема бази даних 30
3.4 Інтеграція з RAWG API 33
3.4.1 Формування параметрів запиту та пагінація 34
3.4.2 Створення або оновлення запису гри 34
3.4.3 Оновлення зв’язків «гра — платформа» та «гра — жанр» 35
3.4.4 Завантаження скріншотів і збереження файлів 36
3.5 Інтеграція цін за допомогою API 37
3.5.1 Очищення назв гри у базі даних для пошуку цін. 37
3.5.2 Пошук гри через API 39
3.5.3 Отримання даних ціни та трейлеру гри 39

3.6 Реалізація бізнес-логіки 40
3.6.1 Отримання та фільтрація списку ігор 40
3.6.2 Детальний перегляд гри 42
3.6.3 Додавання ігор в список улюблених та рейтинг 43
3.7 Інтерфейс користувача 46
3.7.1 Структура шаблонів 46
3.7.2 Динамічне підвантаження контенту javascript 50
3.7.3 Карусель медіа-файлів на сторінці детальною інформації 51
3.8 Реєстрація, вхід та профіль користувача 54
3.9 Функціональні та нефункціональні вимоги 56
3.10 Ручне тестування інтерфейсу 58
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62

ВСТУП

Актуальність теми

У сучасному інформаційному суспільстві значна кількість відеоігор як цифрових продуктів щороку поповнюється сотнями нових проєктів різних жанрів та платформ.

Кожна ігрова платформа має власні маркетплейси зі специфічною ціновою політикою, розпродажами та регіональними знижками, що створює для кінцевого користувача складнощі у порівнянні пропозицій і виборі оптимальної покупки.

Відсутність єдиного зручного інструменту для аналізу та порівняння відеоігор призводить до зайвих витрат, часу та коштів, а також знижує задоволеність від придбання ігор.

Технологічний розвиток і широке поширення інтернету дозволяють реалізувати веб-сервіси, які автоматизовано збирають інформацію з різних джерел через API.

Зокрема, інтеграція з RAWG.io дає змогу отримувати актуальні дані про ігри: описи, скріншоти, трейлери, системні вимоги та поточні ціни. Це відкриває можливості для створення гнучких механізмів фільтрації, сортування та персоналізованих рекомендацій, що значно покращує якість прийняття рішення користувачами.

Розробка веб-сайту GameZone для пошуку та порівняння ігор за жанрам, платформами та цінами є нагальною задачею.

Запропонований сервіс не лише спростить процес вибору найвигідніших пропозицій, але й слугуватиме аналітичним інструментом для відстеження цінових тенденцій у галузі, що важливо як для кінцевих користувачів, так і для бізнесів, які прагнуть оптимізувати свою цінову політику та залучити нових клієнтів через конкурентоспроможні пропозиції

Мета дослідження

Створення зручного та ефективного веб-сайту для пошуку та порівняння відеоігор за ключовими критеріями: жанрами, платформами та цінами.

Завдання дослідження

Розглянути існуючи аналітичні порівняльні сервіси цифрових продуктів із фокусом на відеоігри. Проаналізувати сучасні підходи та технології розробки веб додатків із функціоналом інтелектуального пошуку, фільтрації та сортування даних.

Розробити та реалізувати веб-сайт GameZone з ключовим функціоналом.

Об’єкт дослідження

Процес розробки веб-сайту для пошуку та аналізу відеоігор як інформаційної системи.

Предмет дослідження

Предметом дослідження є технології та методи створення веб-додатків для порівняльного аналізу відеоігор та полегшення кінцевому користувачу вибору ігор та заощадження коштів та часу.

Методи дослідження

Теоретичні: аналіз літературних джерел та порівняння технологій створення інформаційних систем, порівняльний аналіз вибору програмного забезпечення.

Емпіричні: розробка програмного забезпечення на базі фреймворку Django,[6] інтеграція з API RAWG, тестування функціоналу.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому щоб користувачі мали змогу швидко знаходити та порівнювати відеоігри, економлячи час і ресурси.

Отримані рішення можуть бути використані для подальшого розширення функціоналу — додавання нових платформ, сервісів онлайн-магазинів та аналітичних модулів для глибшого дослідження ринку цифрових продуктів.

Глосарій

Платформа — це середовище, де можна запускати відеоігри.

Жанр — тип або категорія гри, що визначає її стиль та ігрові механіки.

API — набір чітко визначених правил, завдяки яким різні частини програмного забезпечення можуть обмінюватися даними. У GameZone API використовується для отримання інформації про ігри з ресурсу RAWG.io.

RAWG — це веб-ресурс та відповідний API, який збирає й надає інформацію про відеоігри, таку як жанри, опис, трейлери, ціни та системні вимоги.

Бекенд — внутрішня частина системи, невидима для користувача. Саме тут відбувається вся обробка інформації, взаємодія з базою даних, опрацювання запитів користувачів та забезпечення логіки роботи всього застосунку.

Фронтенд — зовнішня частина системи, з якою напряму працює користувач. Це веб-інтерфейс, що складається з різноманітних елементів: сторінок, кнопок, форм, меню тощо.

Фільтрація — це відбір даних за певними умовами. Наприклад, показати лише ігри конкретного жанру або певного діапазону цін.

Персоналізація — підбір інформації та налаштування під конкретного користувача, наприклад створення списку ігор, які йому подобаються або можуть бути цікаві.

Інформаційна система — це сукупність технічних та програмних компонентів, які збирають, обробляють, зберігають та показують інформацію.

Веб-застосунок — це програмний продукт, який запускається через веб-браузер. Його не потрібно встановлювати окремо, достатньо зайти на сайт.

Користувач — людина, яка взаємодіє з веб-застосунком для отримання інформації або виконання певних дій.

Django — веб-фреймворк на мові програмування Python, який дозволяє швидко та зручно створювати веб-сайти завдяки вбудованим механізмам роботи з даними, користувачами, шаблонами сторінок та іншим необхідним інструментам.

Модель — це клас у Django, який відповідає таблиці у базі даних. Він визначає, які саме дані зберігати і як з ними взаємодіяти.

Шаблон — це HTML-сторінка з додатковими спеціальними вставками, які Django замінює динамічними даними.

URL-маршрутизація — це механізм у Django, який за адресою в браузері визначає, яка функція чи клас має опрацювати запит та повернути відповідь користувачу.

View — це частина Django-застосунку, яка обробляє запити від користувачів, отримує необхідні дані з бази даних і повертає готову сторінку або інший результат.

База даних — місце, де зберігається вся структурована інформація веб-застосунку. Django зазвичай використовує SQLite для тестування, але для реальної роботи краще підходять PostgreSQL або MySQL.

ForeignKey — це тип зв'язку між двома моделями у базі даних, коли запис в одній таблиці може посилатися на конкретний запис в іншій.

Поле моделі — це конкретна колонка в таблиці бази даних, яка зберігає певний тип інформації.

CRUD — базові операції з інформацією: створення, читання, оновлення та видалення. У веб-додатку ці операції реалізуються через інтерфейс і форми взаємодії з базою даних.

HTTP — протокол, завдяки якому браузер та сервер спілкуються між собою: браузер може отримувати інформацію або надсилати її.

Пагінація — поділ великого списку результатів на сторінки, щоб було зручніше переглядати інформацію.

Кешування — тимчасове зберігання даних, які часто використовуються, щоб прискорити роботу веб-застосунку і не звертатися постійно до зовнішніх сервісів чи бази даних.

Міграції — автоматичний механізм Django, який дозволяє легко змінювати структуру бази даних під час розвитку проєкту.

Система рейтингів — це можливість користувачам виставляти оцінки для відеоігор за визначеною шкалою.

Веб-сервер — програмне забезпечення, яке отримує запити від браузерів користувачів і передає їх веб-застосунку, після чого повертає готові відповіді користувачеві.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Питання порівняння цін та характеристик відеоігор активно розглядається в сучасних джерелах. Публікації присвячені оглядам сервісів, що агрегують інформацію про ігри з різних маркетплейсів, та методикам аналізу їхніх даних. Окрему увагу приділено функціоналу API платформ на кшталт RAWG, IsThereAnyDeal, CheapShark — розглянуто формат запитів, особливості зберігання результатів у власній БД, оптимізацію запитів для швидкої обробки великих обсягів даних [10].

З’ясовано, що більшість рекомендаційних систем ґрунтується на REST API та кешуванні відповідей для зменшення затримок. Під час формування пошукового запиту рекомендується поєднувати фільтрацію за жанрами й платформами з динамічним сортуванням за ціною чи рейтингом.

Внаслідок цього підкреслено, що опрацьовані виявлені підходи можуть бути адаптовані для локальної системи GameZone. Слід звернути увагу на архітектурні шаблони MVC, які забезпечують чітку роздільність між логікою обробки запитів, представленням даних і моделлю збереження результатів.

Під час підготовки першого розділу було виявлено брак робіт, що розкривають реальне поєднання кількох джерел даних у єдиному інтерфейсі, зокрема можливість масового імпорту через адмінку та автоматичне оновлення цін. Існуючі статті радять спиратися на subquery та агрегації у Django ORM для видачі останніх значень цін усіх платформ відразу.

1.2 Аналіз програмного забезпечення

Серед популярних платформ для порівняння цін відеоігор відзначені IsThereAnyDeal, CheapShark, PriceCharting. Усі вони надають широкий вибір фільтрів і механізмів оцінювання, але не дозволяють одночасно контролювати

локальні переваги користувача — наприклад, збереження улюблених ігор та встановлення власного рейтингу.

1.2.1 IsThereAnyDeal

Реалізує агрегування пропозицій із основних цифрових магазинів (Steam, GOG, Epic Games Store, Humble Bundle). Доступна історія зміни цін,[2] налаштування сповіщень про зниження вартості та порівняння регіональних варіантів. При цьому для отримання більшості інформації стає доступними лише після оформлення преміум-підписки. CheapShark забезпечує швидкий доступ до бази Steam-цін, проте обмежується лише ПК-платформою.

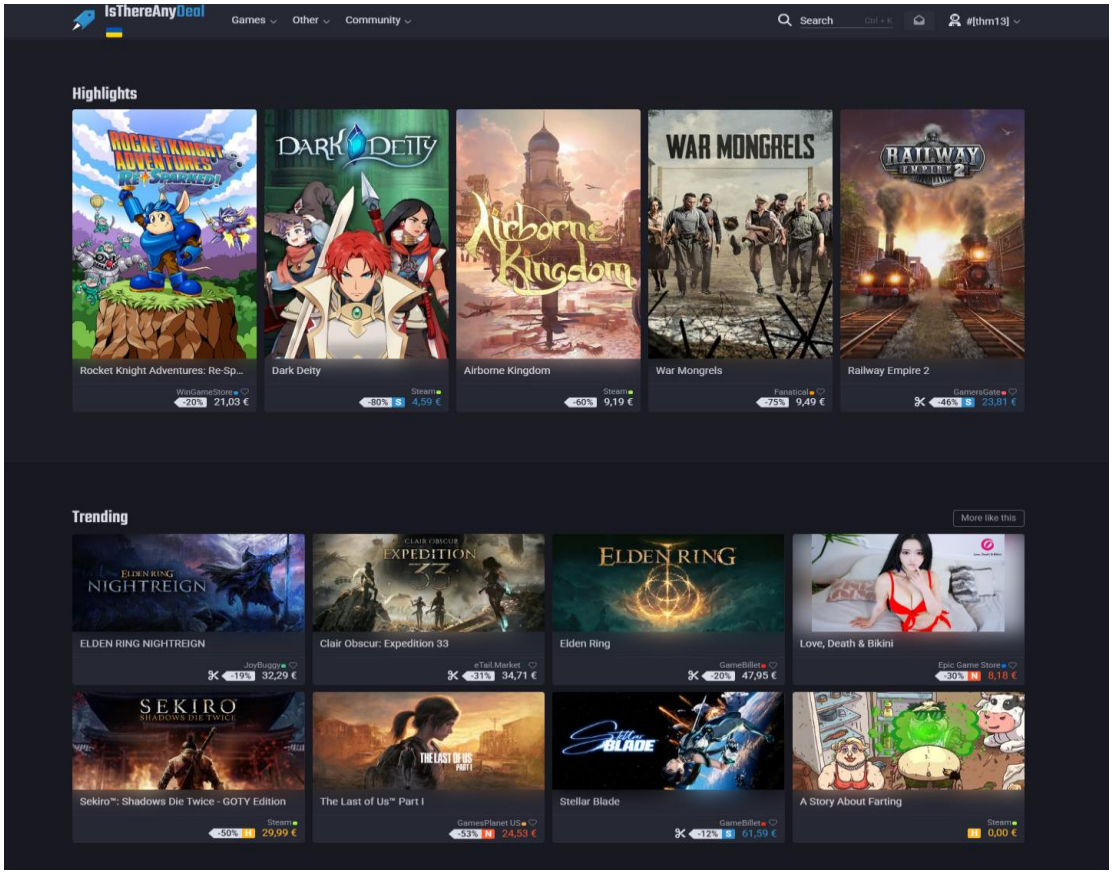


Рисунок 1 — *Візуальний вигляд сторінки IsThereAnyDeal*

1.2.2 CheapShark

Спеціалізується переважно на базі Steam: показуються актуальні знижки, історія цін для окремих ігор, можливість фільтрувати за відсотком знижки чи загальною сумою. Відсутня підтримка інших цифрових крамниць, тому обмежено лише ПК-версіями. Доступні розділи Top Deals і Lowest Prices, що спрощує пошук найвигідніших пропозицій серед усіх ігор у Steam.



Рисунок 2 — *Візуальний вигляд сторінки CheapShark*

1.2.3 PriceCharting

Більше орієнтований на фізичні копії та ретро-ігри: зібрано дані про вартість картриджів, боксів та колекційних видань для консолей різних поколінь (NES, SNES, PlayStation). Для сучасних цифрових релізів детальна

інформація не надається. При цьому можна переглянути середньомісячні коливання цін і скласти портфель для колекціонерів на основі історичних даних.

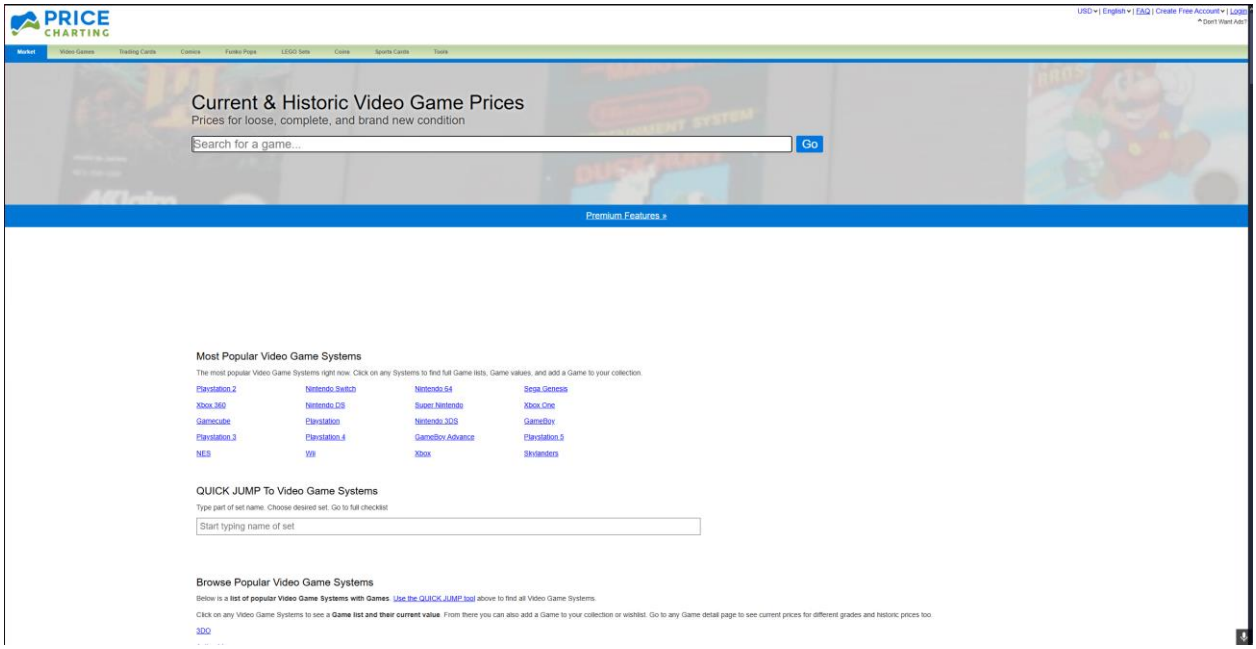


Рисунок 3 — *Візуальний вигляд головної сторінки PriceCharting*

Окрему увагу приділено online-базам: Metacritic, IGDB і RAWG. Metacritic концентрується на зведених рецензіях, IGDB дає детальну інформацію про жанри та дати випуску, але не має механізму порівняння цін. RAWG дозволяє отримувати детальні дані: описи, жанри, платформи, системні вимоги, трейлери, скріншоти, останні ціни в Steam та Epic, при цьому API-ключ надається безкоштовно для некомерційних проєктів.

Проаналізовано, що поєднання можливостей RAWG з власною ORM-модельною структурою у Django спрощує збереження та оновлення інформації. Були розглянуті best practices щодо організації модулів: виділення окремих шарів — data loader, моделі, представлення та шаблони — які гарантують гнучкість і масштабованість.

У результаті було вирішено зробити ставку на RAWG як основне джерело даних. Для забезпечення додаткової надійності передбачено збереження останніх завантажених даних у CSV та можливість оновлювати їх за допомогою окремих керуючих команд. Це дозволяє захиститися від непередбачених збоїв API та створити локальні резервні копії.

1.3 Постановка завдання

Мета кваліфікаційної роботи полягає в розробці веб-застосунку GameZone, що надає можливість швидко знаходити та порівнювати відеоігри за жанром, платформою та ціною, використовуючи API RAWG. Для досягнення поставленої цілі визначено такі ключові завдання:

1. Реалізувати моделі Django для зберігання інформації про ігри, платформи, жанри, ціни, рейтинги та улюблені.
2. Розробити механізм завантаження даних із RAWG API: обробка відповідей, створення/оновлення записів у моделі Game, Platform, Genre; збереження мінімальних і рекомендованих системних вимог українською мовою.
3. Створити сторінку списку ігор із можливістю фільтрації за назвою, жанром, платформою, діапазоном цін та безкоштовними пропозиціями, застосувавши Django-форми і власні шаблонні фільтри.
4. Забезпечити детальний перегляд гри: відображення опису, карусель трейлерів і скріншотів, відображення останньої ціни, середнього рейтингу та персональної оцінки.
5. Реалізувати функціонал реєстрації, входу, профілю користувача згідно зі стандартами Django.
6. Додати можливість додавати/видаляти гру з улюблених та виставляти рейтинги (1–5) через AJAX-запити без перезавантаження сторінки.

7. Налаштувати інтернаціоналізацію (i18n) для мов «укр» та «en», розширивши файл локалізації, щоб весь інтерфейс коректне відображався українською.

8. Розробити адміністративний інтерфейс: що містить кнопку «Імпорт ігор з RAWG», та сторінку для ручного запуску import-процедури.

9. Налаштувати асинхронне оновлення цін через команду для періодичного оновлення моделей Price.

10. Забезпечити адаптивний дизайн шаблонів на Bootstrap 5 з урахуванням темної/світлої теми та перемикача мов, що зчитує вибір із localStorage.

1.4 Висновки до розділу 1

1. Було розглянуті наявні сервіси які частково задовольняють потреби щодо гнучкої фільтрації та можливості персональних рейтингів.

2. Проаналізовано існуючі рішення — IsThereAnyDeal, CheapShark, Metacritic, IGDB, RAWG; обрана платформа RAWG через безкоштовність API, повноту даних і можливість роботи з мовними перекладами.

3. Визначено ключові завдання для реалізації веб-застосунку GameZone: моделі для зберігання інформації, механізми завантаження й відображення даних, функціонал реєстрації й персоналізації.

4. Сформовано чітку архітектурну концепцію: модульність коду, інтернаціоналізація, асинхронне оновлення цін та кешування результатів для пришвидшення робот.

2 ВИМОГИ ДО ОТОЧЕННЯ

2.1 Аналіз сучасних технологій для розробки веб-застосунку

Під час підготовки дипломної було опрацьовано перелік технологій, які оптимально підходять для створення сервісу порівняння відеоігор. Основна мета — забезпечення високошвидкісної обробки запитів, масштабованості та підтримки мультимовності без зайвих зусиль.

Python із фреймворком Django обрав за можливість швидкого запуску прототипу та вбудовані інструменти для ORM-моделювання, авторизації та адміністративної панелі. Django ORM забезпечує зручну роботу з базою даних без потреби писати “чистий” SQL, а система міграцій дає змогу легко змінювати структуру таблиць під час розвитку функціоналу [6].

Для рендерингу інтерфейсу використовується Bootstrap 5 це дозволяє відразу отримати адаптивний дизайн із готовими компонентами та мінімально налаштованим стилем. Застосунок у браузері доповнюється JavaScript (ES6), який реалізує динамічний фільтр, пагінацію й підвантаження даних через AJAX. Це дозволяє відображати список ігор без перезавантаження сторінки [13].

Взаємодія з зовнішнім сервісом RAWG API здійснюється через HTTP-запити, а обмін даними відбувається у форматі JSON. Назви жанрів, опис гри, трейлери, скриншоти та ціни надходять у вигляді структурованих об’єктів, які перетворюються на записи в базі даних. Для періодичного оновлення цін запроваджено керуючу команду update_prices, що автоматично запускається за розкладом.

Переклад описів ігор на українську забезпечує бібліотека googletrans. Використовується кешування результатів, щоб не перевантажувати зовнішнє API. Це дозволяє зекономити час на повторний переклад тих самих фрагментів.

Для зберігання файлів із зображеннями — скриншотами та іконками — передбачено налаштування MEDIA_ROOT і MEDIA_URL у settings.py. У продакшн-режимі передбачено використання Amazon S3 або подібного сховища, що знижує навантаження на локальний сервер.

У цілому перелік застосованих технологій охоплює:

– Python 3.12.6; Django 4.x;

– PostgreSQL 14;

– Bootstrap 5 for CSS;

– JavaScript (ES6) для клієнтської логіки;

– googletrans для перекладів;

– RAWG API для отримання даних про ігри;

– Docker для контейнеризації під час розгортання.

2.2 Аналіз вимог до користувачів веб-застосунку

Головною метою цього аналізу було з’ясування портрету цільової аудиторії та ролі користувачів які користуються застосунком. Основні групи:

1. Гість (неавторизований користувач).

— Можливості: перегляд загального списку ігор із фільтрацією за жанром, платформою, ціною; перегляд детальної сторінки гри

— Обмеження: відсутність функцій додавання до улюбленого чи оцінювання.

2. Зареєстрований користувач.

— Можливості: усе з прав гостя, а додатково – реєстрація облікового запису, вхід у систему, перегляд профілю, додавання/видалення ігор до списку улюбленого, виставлення рейтингу (1–5) без перезавантаження сторінки.

— Вимоги: існування електронної пошти, унікальний логін та пароль, підтверджені стандартною валідацією Django.

3. Адміністратор.

— Можливості: управління усіма моделями; імпорт ігор через кнопку “Імпорт ігор з RAWG” у власному шаблоні; редагування перекладів і локалізованих рядків у файлах; запуск команд через веб-інтерфейс.

— Вимоги: наявність облікового запису з правами superuser; доступ до адмінки лише через захищене з’єднання.

Основні вимоги до інтерфейсу для кожного користувача.

— Інтуїтивна навігація без зайвих кроків між списком ігор та детальною інформацією;

— Зручність фільтрів із мінімізацією натискань: достатньо вибрати жанр, платформу або ввести ключове слово та натиснути Enter;

— Швидкість відгуку: результати фільтрації мають відображатися протягом максимум 300 мс на сучасному середньодоступному обладнанні.

2.3 Аналіз вимог до апаратного та програмного забезпечення

Веб-застосунок орієнтовано на роботу в середовищі розробки. Нижче наведено мінімальні та рекомендовані вимоги.

Мінімальні вимоги до апаратного забезпечення

- Процесор: Intel Core i3 або AMD Ryzen 3.
- ОЗП: 4 GB RAM (для розробки достатньо 8 GB).
- Диск: SSD з вільними 10 GB для розміщення бази даних і медіафайлів.
- Мережеве з’єднання: стабільний інтернет з мінімальною швидкістю 10 Mbps.

Рекомендовані характеристики апаратного забезпечення для продакшн-сервера:

- Процесор: Intel Core i5 чи AMD Ryzen 5.
- ОЗП: 16 GB RAM.
- Диск: RAID-масив з SSD NVMe.

- Мережеве з’єднання: мінімум 100 Mbps з пропускною здатністю до 1 Gbps при пікових навантаженнях.

Програмне забезпечення для середовища розробки:

- ОС: Windows 10 x64, macOS 10.15 або будь-який дистрибутив Linux.
- Python 3.12.6; для управління віртуальним середовищем – pip та virtualenv або Conda.
- Django 4.x зі встановленими залежностями: psycopg2-binary, googletrans.
- PostgreSQL 14.
- Node.js 16.
- Docker 20.x для контейнеризації додатку під час розгортання.
- IDE: VS Code 1.60 або PyCharm.

2.4 Висновки до розділу 2

1. Проведено аналіз сучасних технологій для створення веб-застосунку: Python 3.10, Django 4.x, PostgreSQL, Bootstrap 5, JavaScript (ES6), googletrans, RAWG API, Docker. Виявлено, що цей стек забезпечує необхідну швидкість розробки та масштабованість.

2. Окреслені ролі користувачів: гість, зареєстрований користувач і адміністратор. Установлено, які функції має підтримувати кожна роль, а також визначено вимоги до зручності інтерфейсу й відгуку системи.

3.Описано мінімальні й рекомендовані вимоги до апаратного й програмного забезпечення для середовища розробки та продакшн-серверу. Встановлено, що на базовому обладнанні можлива робота в режимі розробки.

3 РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Опис предметної області

Предметна область охоплює процес отримання, обробки та відображення інформації про відеоігри з різних цифрових платформ задля порівняння цін, жанрів, технічних характеристик та супутніх даних скріншоти, трейлери, рейтинги. Джерелом даних виступає RAWG API, яке агрегує відомості з основних онлайн-магазинів Steam, Epic Games Store, GOG.

У ході роботи дипломної було встановлено, що для кожної відеогри находять структуровані дані у форматі JSON:

— унікальний ідентифікатор та назва;

— текстовий опис із основними особливостями та жанром;

— список платформ ПК, PlayStation, Xbox; із відповідними тегами;

— інформація про розробника та видавця;

— дати виходу (оригінальна та локалізована);

— системні вимоги (мінімальні й рекомендовані);

— URL трейлерів і посилання на скриншоти;

— дані про ціни (поточна вартість у різних магазинах та валютах).

Ці дані зберігаються у реляційній базі (PostgreSQL) із таблицями Game, Platform, Genre, Price, Rating та FavoriteGame. Зв’язки «багато-до-багатьох» між Game і Platform/Genre сприяють гнучкому фільтруванню. Таблиця Price містить історію зміни вартості, що дозволяє відобразити динаміку коливань цін.

Ключові користувацькі сценарії такі:

— гість переглядає каталог ігор, застосовує фільтри (за жанром, платформою, діапазоном цін, безкоштовними релізами) та читає описи;

—зареєстрований користувач може додати гру до списку «Улюблені», виставити оцінку від 1 до 5, переглянути власний профіль з переліком обраного та історією рейтингових оцінок;

—адміністратор у адмінці ініціює імпорт даних із RAWG, керує контентом і контрольовано оновлює інформацію за допомогою команд update_games і update_prices.

Інтерфейс фронтенду реалізовано на Bootstrap 5 і JavaScript (ES6), що забезпечує адаптивність під різні пристрої та динамічну фільтрацію через AJAX, без перезавантаження сторінки. Підтримка локалізації досягається завдяки файлам перекладу Django та бібіліотеці googletrans для автоматичного перекладу описів.

Усередині системи передбачено механізм кешування результатів перекладу та відкладене завантаження зображень, щоб зменшити затримки при завантаженні списку ігор. Фоновий процес оновлення цін дозволяє підтримувати базу даних актуальною без участі кінцевого користувача, а кнопка «Імпорт ігор з RAWG» у власному шаблоні адмінки спрощує ручну перевірку нових релізів.

Предметна область охоплює як технічні аспекти, так і користувацькі вимоги. У результаті з’являється єдина система що дозволяє порівняти відеоігри за різними ознаками й отримати інформацію про найвигідніші пропозиції, інтегруючи дані з декількох джерел в одному веб-інтерфейсі.

3.2 Архітектура системи

Архітектура веб-застосунку представлена двома схемами: компонентною та розгортання. Для розуміння рівнів відповідальності та зв'язки між модулями [3].

3.2.1 Компонентна діаграма

На рисунку 4 показано логічні блоки системи та їх взаємодії, які використовуються в проєкті.

Цей шар надсилає запити до бекенду та відображає отримані дані.

—Application Layer (Backend) включає Django Views, бізнес-логіку і management-команди. Саме тут обробляються запити з фронтенду та реалізується імпорт чи оновлення даних із RAWG API.

—Data Layer (Data_Access) відповідає за роботу з ORM і міграціями, що формують базу даних.

—Infrastructure(Persistence) забезпечує збереження в PostgreSQL, Redis Cache для кешованих результатів і S3-сховище для медіа файлів (скріншотів, трейлерів).

—External Services (RAWG API) виступає зовнішнім джерелом даних, до якого бекенд звертається через HTTP-запити.

На рисунку 5 наведено топологію компонентів у продакшн-середовищі.

— Browser підключається до Nginx через HTTPS.

— Nginx виконує SSL-термінацію й передає запити Gunicorn через WSGI

—Gunicorn запускає процеси DjangoApp, які обробляють запити.

—DjangoApp взаємодіє з PostgreSQL, Redis та S3

Цей підхід гарантує:

- Захищеність каналів зв’язку;
- Розподіл навантаженість між Nginx і Gunicorn;
- Відмовостійкість завдяки кешуванню в Redis і зовнішньому сховищу для великих файлів.

У результаті поєднання компонентної та розгорнутої схем відображається цілісна картина архітектури, яка охоплює внутрішній устрій застосунку, так і фізичне розгортання в продакшен-середовищі.

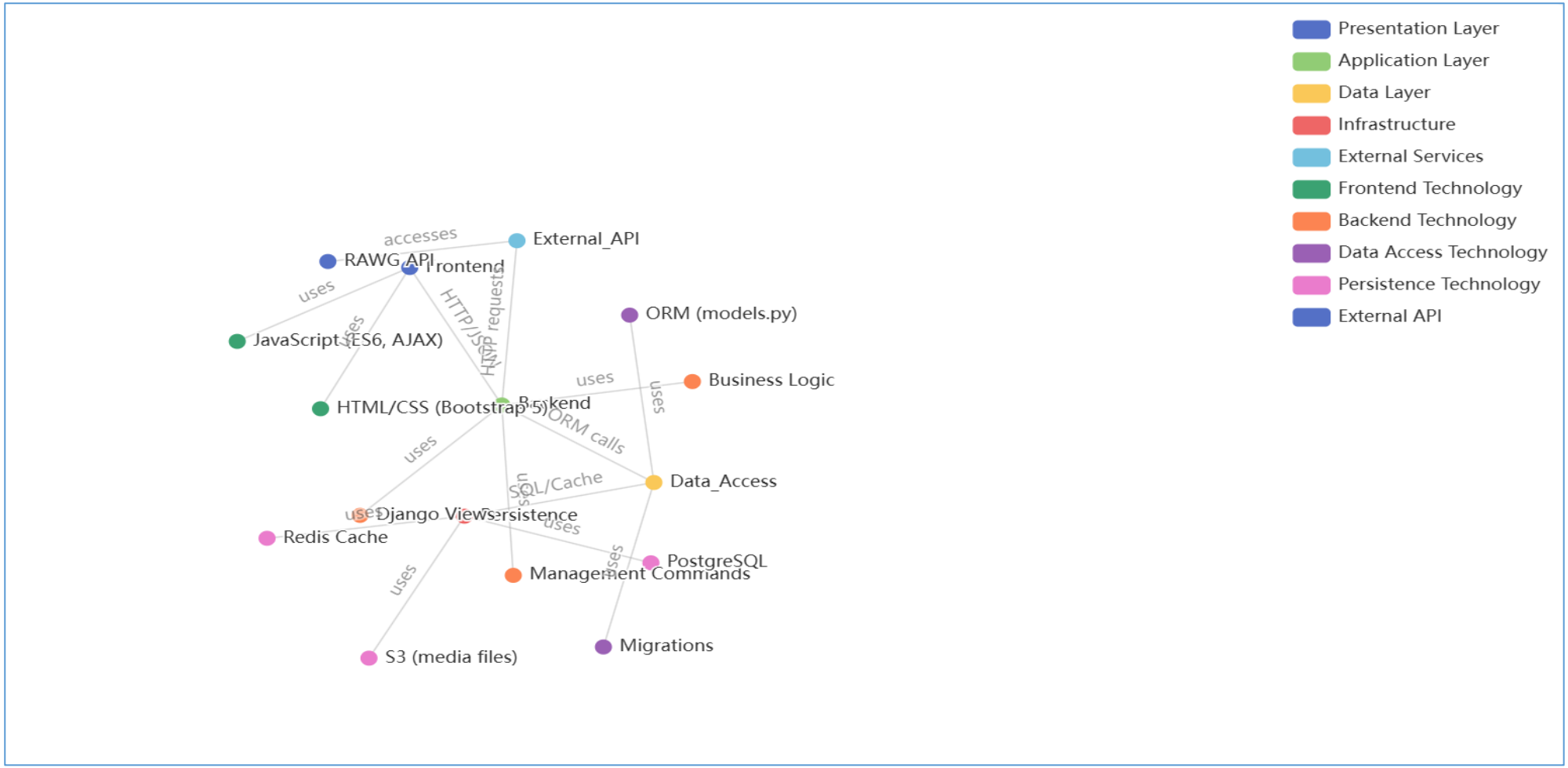
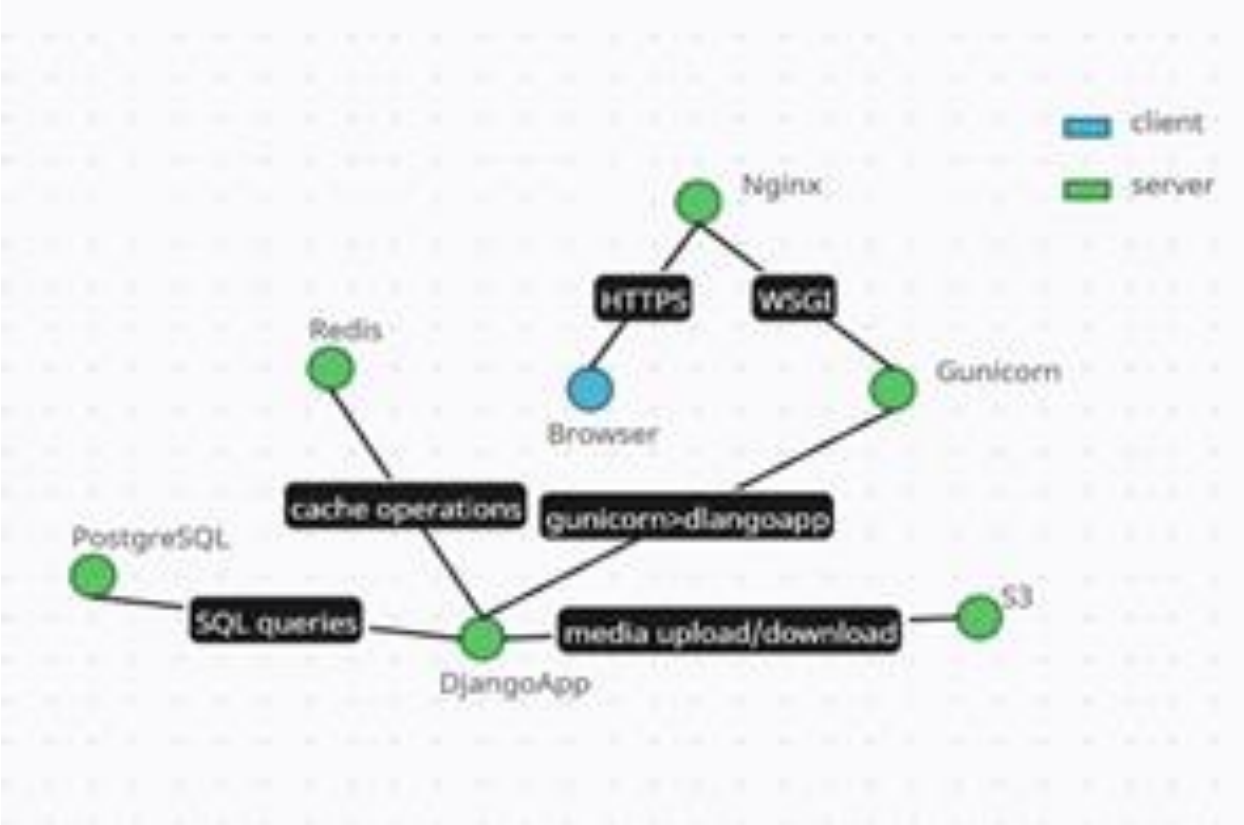


Рисунок 4 — *Компонентна діаграма*

— Presentation Layer(Frontend) сформовано за допомогою HTML/CSS (Bootstrap5) і JavaScript.

3.2.2 Діаграма розгортання

На рисунку 5 наведено топологію компонентів у продакшн-середовищі.



Рисунку 5 — *Відображає діаграму розгортання*

3.2.3 Опис Use-case діаграми

Use-Case діаграма яка зображена на рисунку 6 візуалізує основні сценарії взаємодії трьох категорій акторів із системою GameZone. В центрі уваги — перелік функціональних можливостей, доступних кожній ролі, та межі відповідальності.

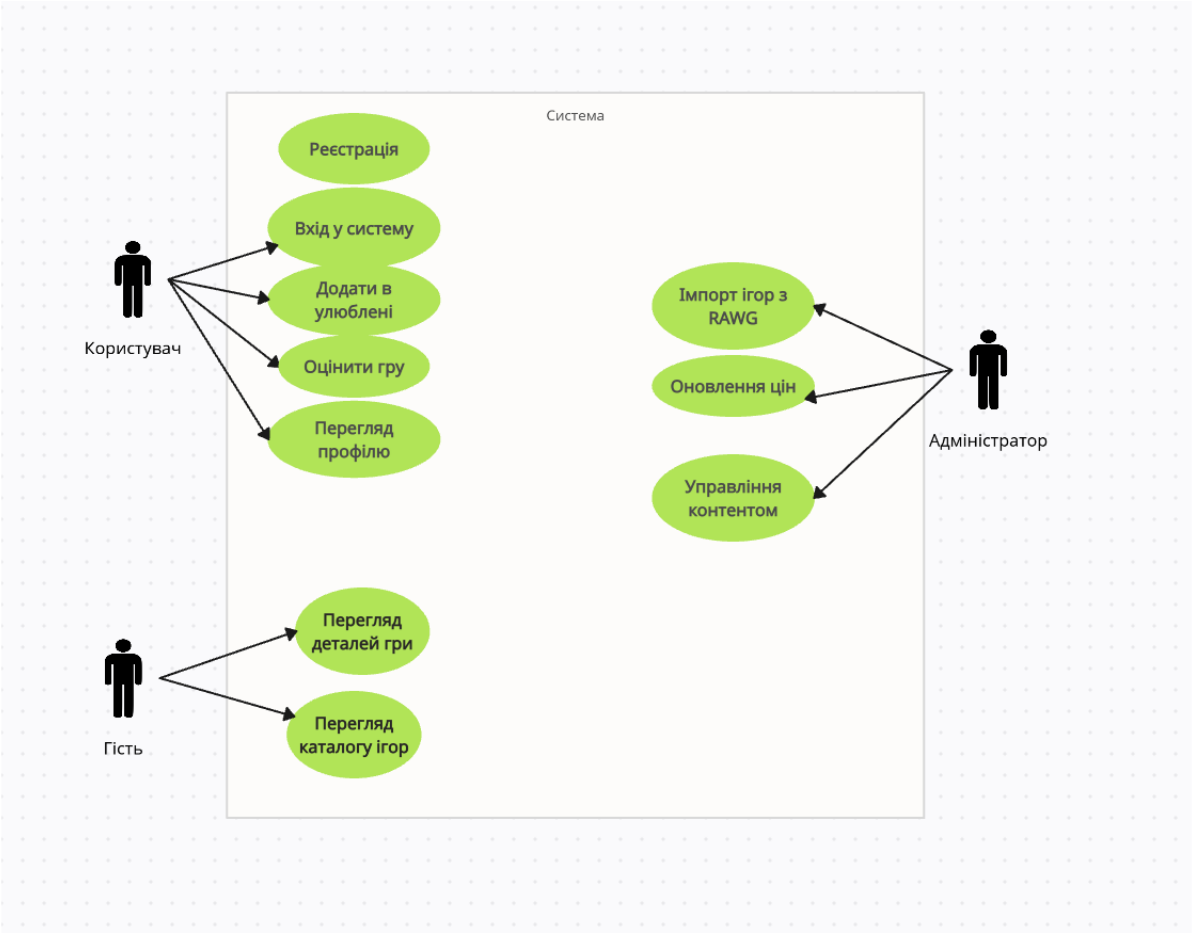


Рисунок 6 — *Відображає Use-Case діаграму*

Також була створена таблиця прецедентів використання системи GameZone. Кожен прецедент описує початкову дію очікуваний результат і роль користувача. Тож таким чином ми можемо перевірити повноту вимог що для кожної дії реалізований відповідний інтерфейс .

Таблиця 1 — *Опис прецедентів*

№	Прецедент	Актор	Короткий опис
1	**2**	**3**	**4**
1	Перегляд каталогу ігор	Гість	Відображення списку ігор із фільтрацією за жанром, платформою та діапазон цін.
2	Перегляд деталей гри	Гість	Відображення опису, трейлерів, скріншотів, цін і середнього рейтингу обраної гри.
3	Реєстрація	Користувач	Створення облікового запису з унікальним логіном та паролем для подальшої взаємодії
4	Вхід у систему	Користувач	Аутентифікацію через форму входу та встановлення сесії для захищеного доступу до персональних функцій.
5	Додати в «Улюблені»	Користувач	Виконання AJAX-запиту для збереження гри у списку улюблених без перезавантаження сторінки.
6	Оцінити гру	Користувач	Встановлення оцінки від 1 до 5 з автоматичним перерахунком середнього рейтингу гри в базі даних.

Продовження таблиці *1 — Опис прецедентів*

1	2	3	4
7	Перегляд профілю	Користувач	Відображення власного списку улюблених ігор та історії виставлених оцінок у профілі користувача.
8	Імпорт ігор з RAWG	Адміністратор	Ініціація команди імпорту даних із RAWG API через соціальну кнопку в адмінці для додавання нових записів.
9	Оновлення цін	Адміністратор	Запуск фонового процесу для періодичного оновлення поточних цін у таблиці Price через команду.
10	Управління контентом	Адміністратор	Додавання, редагування та видалення записів у всіх моделях через стандартний інтерфейс адмінки Django.

Внаслідок опису прецедентів отримано чітке уявлення про сценарії взаємодії та межі відповідальності кожної ролі що допомагає краще розуміти налаштування маршрутів та реалізації класів з контролем доступу та побудови інтерфейсу.

3.3 Схема бази даних

В основі схеми бази даних покладено реляційну модель, вона забезпечує нормалізацію даних до третьої нормальної форми та дозволяє ефективно виконувати фільтрацію й агрегацію. На рисунку 7 зображена схема бази даних яка показує основні сутності та їх зв’язки.

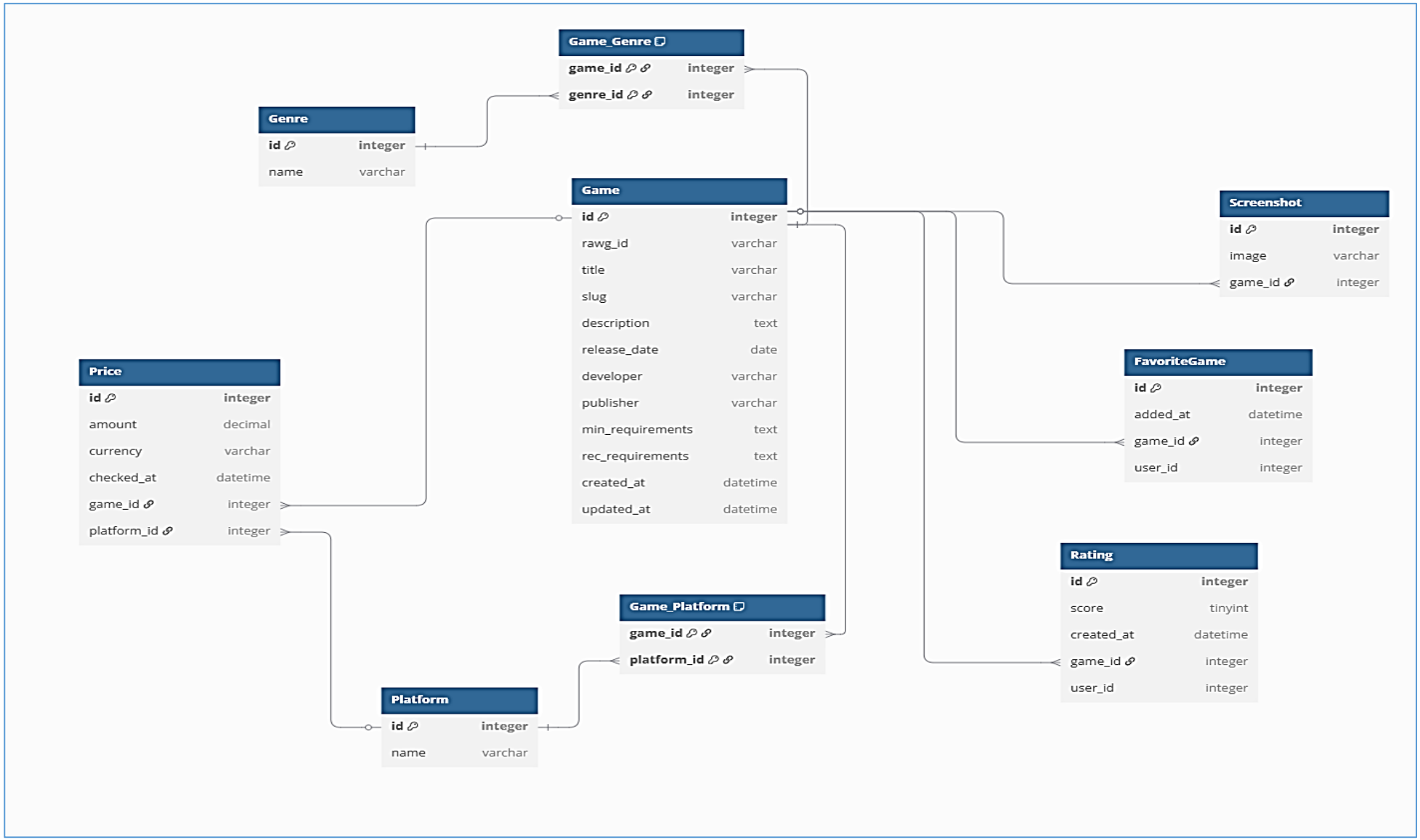


Рисунок 7 — *Схема бази даних*

Game — центральна таблиця, де зберігаються основні атрибути ігор.

- id — унікальний ідентифікатор гри;
- rawg_id — ідентифікатор в зовнішньому API RAWG для синхронізації;
- title — офіційна назва гри;
- slug — URL-фрагмент, створений на основі назви для SEO;
- description — повний текстовий опис;
- release_date — дата першого релізу;
- developer — назва студії-розробника;
- publisher — назва видавця;
- min_requirements — мінімальні системні вимоги;
- rec_requirements — рекомендовані системні вимоги;
- created_at — мітка часу першого запису в БД;
- updated_at — мітка часу останньої зміни запису.

Platform — довідкова таблиця з назвою платформ.

- id — унікальний ідентифікатор платформи;
- name — назва платформи (PC; Xbox, PlayStation);

Genre — довідкова таблиця з назвою жанрів.

- id — унікальний ідентифікатор жанру;
- name — назва жанру (шутер, RPG, стратегія);

Game_Platform — проміжна таблиця, що реалізує зв’язок «багато до багатьох» між Game і Platform. Комбіновані індекси гарантують унікальність записів.

- game_id — посилання на гру;
- platform_id — посилання на платформу;

Game_Genre — проміжна таблиця, що реалізує зв’язок «багато до багатьох» між Game і Genre. Комбіновані індекси гарантують унікальність записів.

- game_id — посилання на гру;
- genre_id — посилання на жанр;

Screenshot — зображення гри; кожен запис належить лише одній грі.

- id — унікальний ідентифікатор зображення;
- image — шлях або URL до файлу скриншота;
- game_id — гра, до якої належить зображення.

Price — історія вартості гри на платформі.

- id — унікальна мітка запису ціни;
- amount — числове значення вартості;
- currency — валюта;
- game_id — гра, для якої зафіксовано ціну;
- platform_id — платформа, де отримано вартість.

Rating — оцінка користувача (1-5) із полем user_id для зв’зку з таблицею користувачів Django.

- id — унікальний ідентифікатор оцінки;
- score — значення оцінки від 1 до 5;
- game_id — оцінювана гра;
- user_id — автор оцінки (користувач);

FavoriteGame — список улюблених ігор кожного користувача.

- id — унікальна мітка для запису в обраному;
- game_id — гра, додана в обране;
- user_id — користувач, що додав гру.

В результаті побудована база даних яка забезпечує гнучкість фільтрації, цілісність даних так високу продуктивність запитів для реалізації функціоналу веб-застосунку.

3.4 Інтеграція з RAWG API

Одним із важливих аспектів програми є саме інтеграція ігор за допомогою доволі популярного API який надає доступ до великої кількості ігор з наявністю більшості інформації про доступні ігри окрім ціни та трейлерів

3.4.1 Формування параметрів запиту та пагінація

Для поетапного отримання переліку ігор формується набір *параметрів пошуку* (Лістинг 1) які складаються з фільтрацією за датою релізу, розміром сторінки та платформами[9].

Лістинг 1 — Параметри пошуку

```
params = {
    "key": API_KEY,
    "page": page,
    "page_size": PAGE_SIZE,
    "ordering": "-released",
    "dates": f"{LAST_YEAR_START},{TODAY_DATE}",
    "platforms": ",".join(str(pid) for pid in
ALLOWED_PLATFORM_IDS),

}

resp = requests.get(BASE_LIST_URL, params=params)
resp.raise_for_status()
data = resp.json() or {}
results = data.get("results", [])
```

- API_KEY отримується з налаштувань Django;
- *ordering= «-released»* гарантує пріоритет найсвіжіших релізів;
- dates обмежує вибірку іграми останнього року (LAST_YEAR_START… TODAY_DATE);
- platforms відбирає лише дозволені платформи.

Цикл *while imported < max_games* з інкрементом page дозволяє поступово завантажувати до max_games результатів без одночасного навантаження на API.[14]

3.4.2 Створення або оновлення запису гри

Для того щоб уникнути дублювання записів у базі даних застосовується метод get_or_create, який спочатку *перевіряє наявність гри* за параметром slug, а у разі відсутності — створює новий об’єкт із вказаними властивостями. (Лістинг 2)

Лістинг 2 — Перевірка наявності гри

```
game, created = Game.objects.get_or_create(
    slug=slug,
    defaults={
        "title": title,
        "description": desc,
        "release_date": release_date,
        "developer": developer_name,
        "publisher": publisher_name,
        "min_requirements": min_req,
        "rec_requirements": rec_req,
        "trailer_url": trailer_url,
    }
)
```

- пошук за slug запобігає створенню дубльованих записів;
- при створенні нового об’єкту збільшується лічильник імпортованих ігор;
- у разі існування запису оновлення відбувається тільки для порожніх полів, game.save() викликається лише за необхідності.

За допомогою цієї частини коду забезпечується економне заповнення бази збережено вже вручну відкориговані дані, а нові й потрібні поля автоматично доповнюються.

3.4.3 Оновлення зв’язків «гра — платформа» та «гра — жанр»

Динамічне встановлення зв’язків між сутністю Game та довідковими таблицями Platform і Genre використовують ітерацію по результатах API. Якщо в отриманих даних знаходиться нова платформа чи жанр, відповідний запис створюється автоматично, після чого через M2M-поле відбувається прив’язка до гри. (Лістинг 3)

Лістинг 3 — Прив’язка жанру чи платформи до гри

```
for pd in platforms_list:
    pname = pd.get("platform", {}).get("name")
    if pname:
```

```
plat_obj, _ = Platform.objects.get_or_create(name=pname)
        game.platforms.add(plat_obj)

for gd in det.get("genres", []):
    gname = gd.get("name")
    if gname:
        genre_obj, _ =
Genre.objects.get_or_create(name=gname)
        game.genres.add(genre_obj)
```

— *get_or_create* створює новий запис платформи та жанру лише за відсутності такого в базі;

— game.platforms.add(plat_obj) і game.genres.add(genre_obj) додають відношення у приховані проміжні таблиці games_game_platforms і games_game_genres;

— оновлені зв’язки стають одразу доступними для фільтрації та сортування без додаткових SQL-запитів.

Завдяки цьому підходу додається гнучкість: нові платформи й жанри впроваджуються без змін моделі, а існуючі ігри не втрачають коректних відношень.

3.4.4 Завантаження скріншотів і збереження файлів

Автоматизація завантаження та *збереження скріншотів* (Лістинг 4) реалізовано окремий запит до ендпоінту RAWG, після чого файли зберігаються безпосередньо в медіа-сховище Django.

Лістинг 4 — Завантаження скріншотів

```
ss_resp =
requests.get(f"{BASE_DETAIL_URL}/{item['id']}/screenshots",
params={"key": API_KEY})
if ss_resp.status_code == 200:
    ss_data = ss_resp.json() or {}
    for shot in ss_data.get("results", []):
        img_url = shot.get("image")
        if not img_url:
            continue
```

```
r2 = requests.get(img_url)
if r2.ok:
    img_name = img_url.split("/")[-1]
    shot_obj, created_ss =
Screenshot.objects.get_or_create(game=game, image=img_name)
    if created_ss:
        shot_obj.image.save(img_name,
ContentFile(r2.content), save=True)
```

— виконано запит до скріншотів, який повертає масив URL зображень

— при відсутності поля image запис пропускається, що запобігає помилкам порожніх даних

— HTTP-відповідь r2.ok гарантує отримання коректного контенту;

— метод get_or_create уникає дублювання скріншотів у базі;

— прапорець save=True забезпечує одночасне збереження моделі та файлу, а наступні виклики ігнорують вже збережені зображення.

Застосування цього механізму гарантує уникнення повторних завантажень тих самих ресурсів, зменшує надлишкові запити до RAWG і водночас забезпечує доступність скриншотів через медіа без додаткової обробки.

3.5 Інтеграція цін за допомогою API

В силу того що за допомогою RAWG не можливо було передати інформацію про ціни, використовувалась ще одна API для оновлення цін усіх ігор які знаходяться у базі даних [1].

3.5.1 Очищення назв гри у базі даних для пошуку цін.

Спочатку для того щоб API працювало правильно ми прибираємо з назв гри усі коми, усі знаки які можуть в теорії заважати пошуку цін та трейлерів по назві (Лістинг 5)[15].

Лістинг 5 — Очищення назви гри

```
def clean_title_for_steam_search(raw_title: str) -> str:
    title = raw_title.strip()
# 1) Обрізка після перших роздільників
    for sep in [":", "-", "—", "(", "["]:
        if sep in title:
            title = title.split(sep, 1)[0].strip()

    # 2) Заміна римських цифр на арабські в кінці
    roman_match   =   re.search(r"\b([IVXLCM]+)\b$",   title,
re.IGNORECASE)
    if roman_match:
        roman = roman_match.group(1).upper()
        roman_to_arabic                                    =
{"I":"1","II":"2","III":"3","IV":"4","V":"5","VI":"6","VII"
:"7","VIII":"8","IX":"9","X":"10"}
        if roman in roman_to_arabic:
            title          =          re.sub(r"[IVXLCM]+$",
roman_to_arabic[roman], title, flags=re.IGNORECASE)

    # 3) Видалення надлишкових слів (видання, доповнення)
    patterns_to_remove = [
        r"\bComplete       Edition\b",       r"\bDeluxe\b",
r"\bRemastered\b",
        r"\bGame of the Year\b", r"\bAnniversary Edition\b",
r"\bHD\b",
        r"\bTrilogy\b", r"\b(19|20)\d{2}\b"
    ]
    for pat in patterns_to_remove:
        title      =      re.sub(pat,      "",      title,
flags=re.IGNORECASE).strip()

    # 4) Очищення зайвих пробілів
    return re.sub(r"\s{2,}", " ", title).strip()
```

— Крок 1: відсікаються частини після символів : – — ([, щоб уникнути некоректних розширень у пошуковому запиті;

— Крок 2: дозволяє коректно обробляти назви зі римськими цифрами;

— Крок 3: видаляються повторювані уточнення типу “Deluxe”, “Game of the Year” тощо, що зменшує кількість невдалих пошуків;

— Крок 4: оптимізує формат рядка перед передачею в API;

3.5.2 Пошук гри через API

Після того як ми оптимізували назву під можливий пошук через API *розпочинаємо пошук* (Лістинг 6), у нас є URL посилання на яке ми подаємо запит з певними параметрами та з невеликим тайм-аутом щоб наш запит не зависав. Також може відбутися пропуск якщо пошук не повернув жодного id.

Лістинг 6 — Пошук гри

```
search_url ="https://store.steampowered.com/api/storesearch"
params_search = {"term": search_title, "l": "english", "cc":
"US", "size": 1}

resp   =   requests.get(search_url,   params=params_search,
timeout=10)
resp.raise_for_status()
products = resp.json().get("products", [])
if not products:
    continue

appid = products[0].get("id")
```

3.5.3 Отримання даних ціни та трейлеру гри

Отримавши результат ми перевіряємо success що дані повернено коректно, якщо ігри безкоштовні то наш ключ is_free повертає значення True що облегшує пошук і зайві записи, також отримуємо ціну у центах.(Лістинг 7)

Лістинг 7 — Отримання інформації

```
details_url =
"https://store.steampowered.com/api/appdetails"
params_details = {
    "appids": appid,
    "cc": "US",
    "l": "english",
}
try:
    resp2 = requests.get(details_url,
params=params_details, timeout=10)
    resp2.raise_for_status()
info_wrapper = details_data.get(app_key, {})
if not info_wrapper.get("success"):
continue
```

```
info = info_wrapper["data"]
if info.get("is_free"):
continue
```

3.5.4 Розрахунок ціни та створення запису

Перетворюємо центи у долари через Decimal що забезпечує точність фінансових обчислень. За наявності знижки автоматично відображає діючу вартість.(Лістинг 8)[15]

Лістинг 8 — Створення запису

```
initial_cents = po.get("initial")
final_cents = po.get("final")
if initial_cents is None or final_cents is None:
    raise ValueError("Нет полей initial/final")
retail_price = Decimal(initial_cents) / Decimal(100)
final_price = Decimal(final_cents) / Decimal(100)
discount_pct = po.get("discount_percent", 0)
if discount_pct and discount_pct > 0:
    use_price = final_price
    note = f" (скидка {discount_pct}%)"
Price.objects.create(
    game=game,
    platform=platform_obj,
    price=use_price,
    date_checked=timezone.now().date()
)
```

3.6 Реалізація бізнес-логіки

Розглянемо ключові view-функції та класи, що використовуються для візуалізації поведінки веб-застосунку: відображення списку ігор, деталізацію, AJAX-обробку улюблених ігор та рейтингу.

3.6.1 Отримання та фільтрація списку ігор

Для відображення каталогу з можливістю пошуку та фільтрації (Лістинг 9) використовується функція game_list. Вона формує запит до бази даних,

поступово застосовуючи критерії — ті які завад користувач після чого показує результат у вигляді карток із актуальною інформацією про ціни та з можливістю відмітити свою улюблену гру.[14]

Лістинг 9 — Отримання та фільтрація ігор

```
def game_list(request):
    qs = Game.objects.all()
    # Пошук по назві
    q = request.GET.get("q")
    if q:
        qs = qs.filter(title__icontains=q)

    # Фільтри жанрів і платформ
    genres = request.GET.getlist("genre")
    platforms = request.GET.getlist("platform")
    if genres:
        qs = qs.filter(genres__id__in=genres)
    if platforms:
        qs = qs.filter(platforms__id__in=platforms)

    # Діапазон цін
    min_price = request.GET.get("min_price")
    max_price = request.GET.get("max_price")
    if min_price or max_price:
        qs = qs.filter(prices__amount__gte=min_price or 0,
                       prices__amount__lte=max_price or 1e9)

    # Анотація останньої ціни та пагінація
    latest_price = Subquery(
        Price.objects
             .filter(game=OuterRef('pk'))
             .order_by('-checked_at')
             .values('amount')[:1]
    )
    qs = qs.annotate(current_price=latest_price).distinct()
    page = request.GET.get("page", 1)
    paginator = Paginator(qs, 12)
    games = paginator.get_page(page)

    # Мітки улюбленого для зареєстрованого
    fav_ids = []
    if request.user.is_authenticated:
        fav_ids                                              =
request.user.favoritegame_set.values_list('game_id',
flat=True)
```

```
    return render(request, "games/game_list.html", {
        "games": games,
        "fav_ids": set(fav_ids),
    })
```

—фільтрування через GET-параметри відбувається послідовно: пошук за ключовим словом, відбір за жанрами та платформами, обмеження за мінімальною та максимальною ціною;

—Subquery підтягує єдину останню ціну для кожної гри без виконання додаткових складних JOIN-запитів;

—Paginator ділить отриманий набір на сторінки по 12 об’єктів, що полегшує навігацію;

—для авторизованих користувачів збирається множина ідентифікаторів

3.6.2 Детальний перегляд гри

Для формування сторінки з детальною інформацією (Лістинг 10) про обрану гру відповідає функція game_detail. Вона збирає усі можливі дані такі як: опис гри, ціна, рейтинг, особиста оцінка та їх підготовку для рендерингу в шаблоні [12].

Лістинг 10 — Створення сторінки детального перегляду гри

```
def game_detail(request, pk):
    game = get_object_or_404(Game, pk=pk)
    # Остання ціна
    price = game.price_set.order_by('-checked_at').first()

    # Переклад опису для української локалі
    desc = game.description
    if request.LANGUAGE_CODE == "uk":
        desc = translate(desc)

    # Середній рейтинг
    avg                                                    =
game.rating_set.aggregate(avg_score=Avg('score'))['avg_scor
e']

    # Особиста оцінка
```

```
    user_score = None
    if request.user.is_authenticated:
        user_rating                                          =
game.rating_set.filter(user=request.user).first()
        user_score = user_rating.score if user_rating else
None

    return render(request, "games/game_detail.html", {
        "game": game,
        "description": desc,
        "price": price,
        "avg_score": avg,
        "user_score": user_score,
    })
```

— *get_object_or_404* гарантує повернення 404 у разі відсутності гри з вказаним параметром;

— обчислення середнього рейтингу Avg виконується на рівні бази даних для швидкої агрегації;

— персональна оцінка дістається одним запитом до моделі Rating, що знижує кількість звернень до БД;

— у шаблон передається повний контекст з грою, описом, ціною, середнім рейтингом та індивідуальною оцінкою.

3.6.3Додавання ігор в список улюблених та рейтинг

Додавання та оновлення списку улюблених ігор або рейтингу без перезавантаження сторінки (Лістинг 11) застосовується AJAX-підхід у комбінації з Django-view-функціями, що повертають JSON-відповіді.

Лістинг 11 — Додавання у список улюблених ігор та рейтингу

```
@require_POST
def toggle_favorite(request, game_pk):
    game = get_object_or_404(Game, pk=game_pk)
    fav,                        created                      =
FavoriteGame.objects.get_or_create(user=request.user,
game=game)
    if not created:
        fav.delete()
    return JsonResponse({"favorited": created})
```

```
@require_POST
def rate_game(request, game_pk):
    score = int(request.POST.get("score", 0))
    if not 1 <= score <= 5:
        return JsonResponse({"error": "Invalid score"},
status=400)    Rating.objects.update_or_create(
        user=request.user,
        game_id=game_pk,
        defaults={"score": score, "created_at":
timezone.now()}
    )
    avg =
Rating.objects.filter(game_id=game_pk).aggregate(Avg('score
'))['score__avg']
    return JsonResponse({"avg_score": avg})
```

—@require_POST блокує доступ через GET, захищаючи від некоректних запитів;

— toggle_favorite через get_or_create створює запис у FavoriteGame або видаляє його, повертаючи статус у полі favorited;

— rate_game приймає оцінку, перевіряє коректність (1–5), потім оновлює чи створює запис у Rating, після чого повертає новий середній рейтинг у полі avg_score.

Також була створена діаграма послідовності яка показує взаємодію користувача, фронтенду, бекенду та БД, що дає можливість швидше зробити свій запит і синхронний стан інтерфейсу.

Діаграма відображає послідовність дій при обробці AJAX-запитів для таких функцій: список улюблених ігор та рейтингу. Користувач натискаючи на кнопку «Улюблене» або встановлення рейтингу ініціює подію на стороні клієнту. Після чого фронтенд надсилає запит до бекенду де відбувається виклик додавання в улюблені ігри або для додавання рейтингу. Перевіряється чи існує запит якщо ні створюється новий рядок у таблиці, якщо так запис оновлюється у випадку рейтингу або видаляється у випадку списку «Улюблених ігор». Далі формується JSON-відповідь із полем favorited або оновленим середнім рейтингом, фронтенд отримує відповідь та оновлює

іконку відображає нове значення рейтингу не оновлюючи сторінку. Користувач бачить актуальний стан інтерфейсу одразу після виконання запиту.

Дана діаграма послідовності (Рисунок 8) гарантує миттєвий відгук і синхроність даних у UI, зменшуючи навантаження на сервер і покращуючи користувацький досвід.

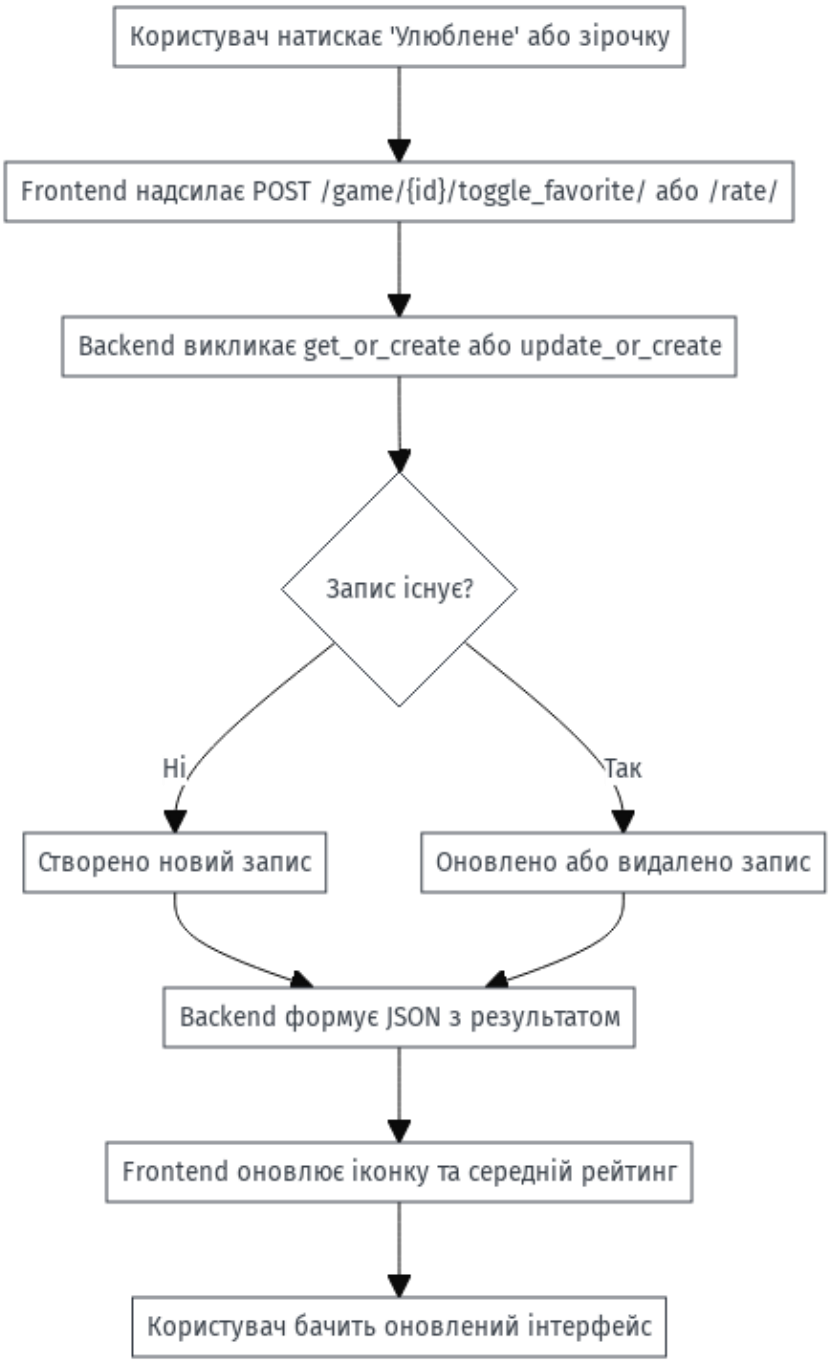


Рисунок 8 — *Відображає діаграму послідовності*

3.7 Інтерфейс користувача

Важливим аспектом роботи веб-застосунку є користувацький інтерфейс, бо головне щоб користувачу було інтуїтивно все зрозуміло, не дуже різкий візуал, щоб він був доволі новим і він захотів користуватися цим застосунком та повертатися на сторінку знову, тому у цьому розділі буде описано як реалізовано клієнтську частину GameZone: шаблони, компоненти фільтрації, картки ігор [13].

3.7.1 Структура шаблонів

Веб-застосунок містить загальний каркас (Лістинг 11) до якого підключені стилі Bootstrap, налічуються кнопки зміни світлого та темного режиму для зручності клієнта, кнопка логотипу сайту яка повертає на головну сторінку зі списком ігор, кнопки перемикача мови українська та англійська і кнопка профілю з регістрацією чи логіну для гостей.

Лістинг 11 — Загальний каркас

```
<!DOCTYPE html>
<html lang="{{ LANGUAGE_CODE }}">
<head>
  …
  <link href="{% static 'css/bootstrap.min.css' %}"
rel="stylesheet">
</head>
<body class="{% if request.COOKIES.theme=='dark' %}dark-
mode{% endif %}">
  <nav class="navbar …">
    <a href="{% url 'game_list' %}">🎮 GameZone</a>
    <button id="theme-toggle">◐</button>
    <div class="lang-switch">
      <a href="/?lang=en">EN</a> | <a
href="/?lang=uk">УК</a>
    </div>
  </nav>
  <main class="container py-4">
    {% block content %}{% endblock %}
  </main>
  {% block scripts %}{% endblock %}
```

Головна сторінка далі складається з шаблону game_list який наслідує загальний каркас з усіма налаштуваннями візуалу сторінки включає в себе блок карток ігор у сітці (Лістинг 12) де відображається ціна, скріншот гри та невелика частина опису з кнопкою додати в список «Улюблених ігор», фільтрації ігор (Лістинг 13) за такими налаштуваннями як: строка пошуку, фільтрація за платформою, жанрами, мінімальною та максимальною ціною ігор а також налаштування премикача показувати тільки безкоштовні ігри.

Лістинг 12 — Відображення картки ігор у сітці

```
<div class="row row-cols-1 row-cols-md-3 g-4">
  {% for game in games %}
    <div class="col">
      <div class="card h-100">
        <img  src="{{  game.screenshot_url  }}"  class="card-
img-top">
        <div class="card-body">
          <h5 class="card-title">{{ game.title }}</h5>
          <p                              class="card-text">{{
game.description|truncatechars:100 }}</p>
        </div>
        <div   class="card-footer   d-flex   justify-content-
between">
          <span>₴{{ game.current_price }}</span>
          {% if user.is_authenticated %}
            <button class="btn fav-btn" data-id="{{ game.id
}}">
              {% if game.id in fav_ids %}♥{% else %}♡{% endif
%}
            </button>
          {% endif %}
        </div>
      </div>
    </div>
  {% endfor %}
</div>
```

Зі сторони користувача інтерфейс виглядає інтуїтивним, гарним, та доволі зручним у використані клієнтом (Рисунок 9).

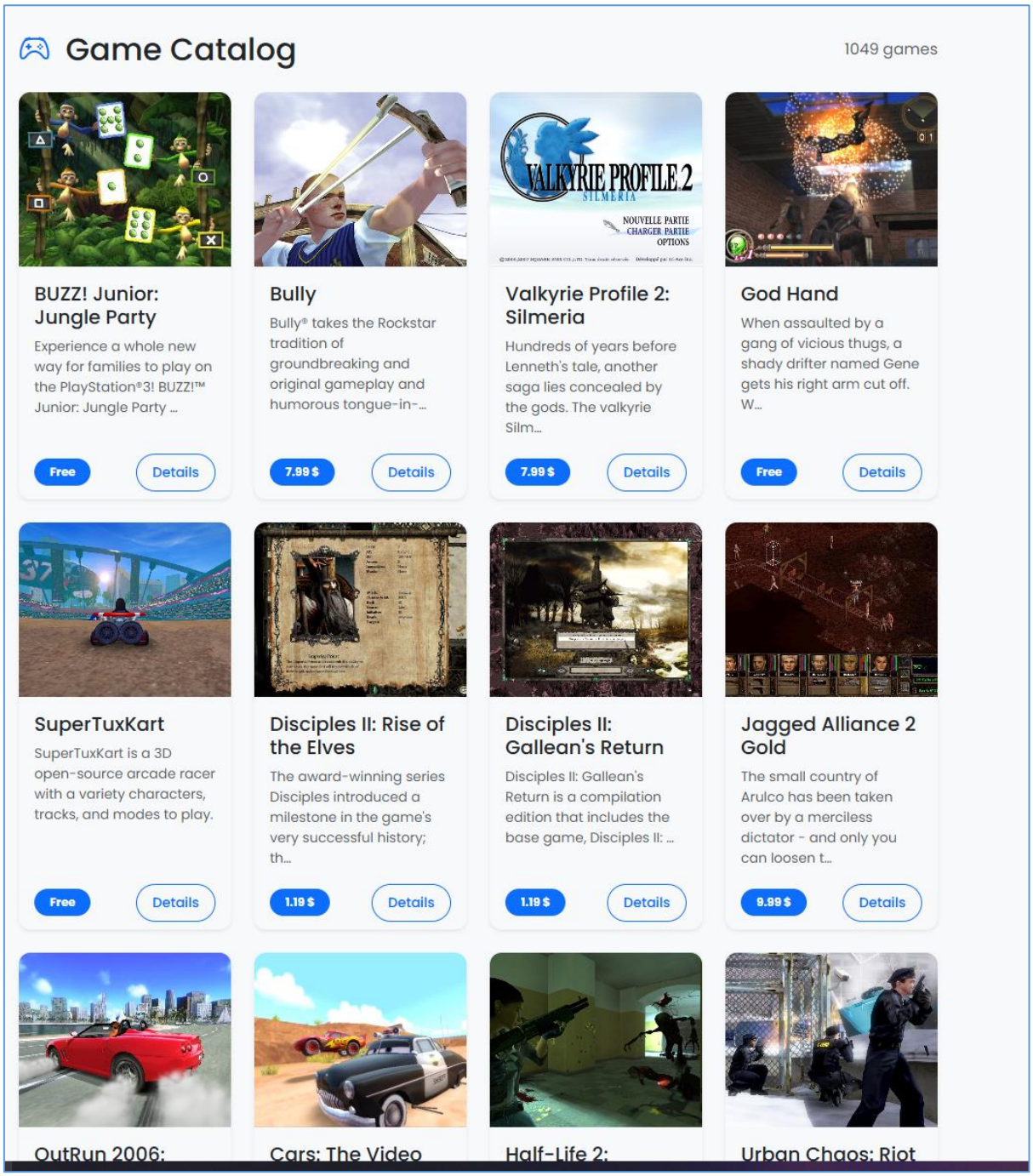


Рисунок 9 — *Результат реалізації карток користувацького інтерфейсу*

Демонстрація реалізації користувацького інтерфейсу фільтрація ігор, яка є одною з базових вимог веб-застосунку.

Лістинг 13 — Відображення реалізації фільтрації ігор

```
<form method="get" class="row g-2 mb-3">
  <div class="col-md-4">
<input name="q" value="{{ request.GET.q }}"
placeholder="Пошук..." class="form-control">
```

```
</div>
  <div class="col-md-2">
    {{ form.genre|add_class:"form-select" }}
  </div>
  <div class="col-md-2">
    {{ form.platform|add_class:"form-select" }}
  </div>
  <div class="col-md-2">
    <input name="min_price" value="{{ request.GET.min_price
}}" placeholder="Від ₴" class="form-control">
  </div>
  <div class="col-md-2">
    <button type="submit" class="btn btn-primary w-
100">Фільтрувати</button>
  </div>
</form>
```

Результат вигляду фільтрації зі сторони користувача (Рисунок 10).

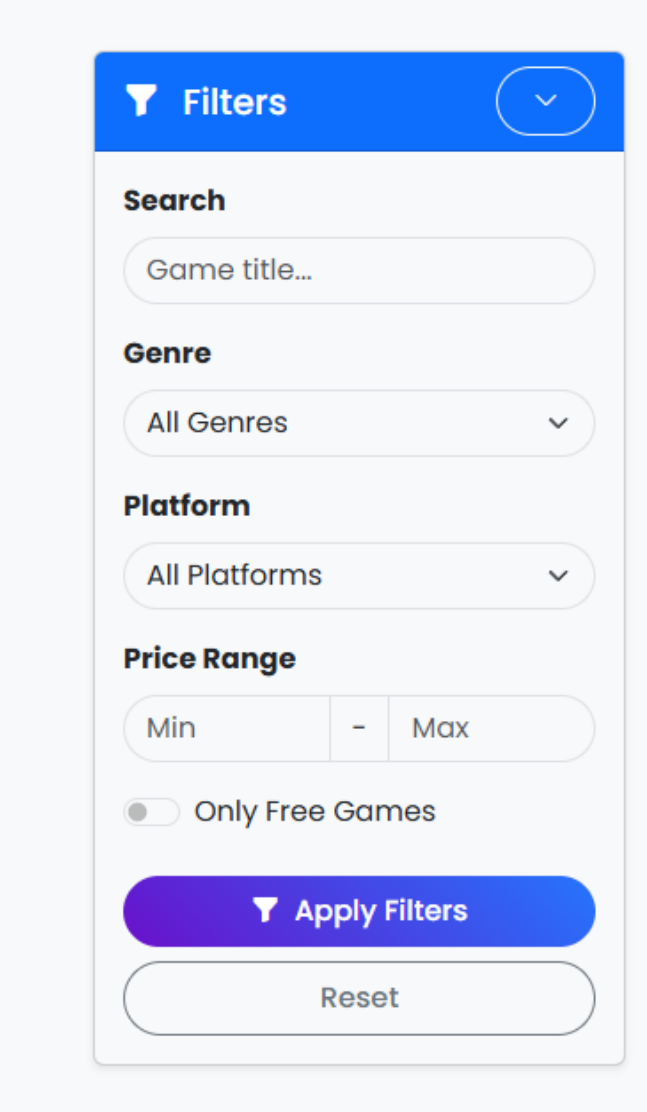


Рисунок 10 — *Результат візуальної реалізації фільтрації*

3.7.2 Динамічне підвантаження контенту javascript

Щоб облегшити роботу сторінки було використано миттєве оновлення інтерфейсу без повного перезавантаження сторінки (Лістинг 14) використовуючи мову програмування JavaScript було застосовано AJAX-запит за допомогою fetch.[8]

Лістинг 14 — Оновлення інтерфейсу без перезавантаження сторінки

```
<script>
  // 1. Знайти всі кнопки «Улюблене»
  document.querySelectorAll('.fav-btn').forEach(btn => {
    // 2. Повісити обробник кліку на кожну
    btn.addEventListener('click', () => {
      const gameId = btn.dataset.id;  // отримання ID гри з
data-атрибута

      // 3. Відправити POST-запит із CSRF-токеном
      fetch(`/game/${gameId}/toggle_favorite/`, {
        method: 'POST',
        headers: {
          'X-CSRFToken': getCookie('csrftoken'),
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({}) // тіло може бути порожнім
      })
      // 4. Обробити відповідь як JSON
      .then(response => {
        if (!response.ok) {
          throw new Error('Не вдалося оновити улюблене');
        }
        return response.json();
      })
      // 5. Оновити текст кнопки відповідно до статусу
      .then(data => {
        btn.textContent = data.favorited ? '♥' : '♡';
        btn.classList.toggle('favorited', data.favorited);
      })
  function getCookie(name) {
    const matches = document.cookie.match(
      new RegExp('(^|; )' + name + '=([^;]*)')
    );
    return matches ? decodeURIComponent(matches[2]) : '';
  }
</script>
```

— усі кнопки з класом *.fav-btn* знаходяться у DOM й отримують обробник кліку.

—*btn.dataset.id* повертає gameId, що відповідає грі, обраному користувачем.

— в заголовку передається X-CSRFToken, отриманий із cookie, щоб Django допустив POST-операцію.

—*.catch()* ловить будь-які винятки й виводить їх у консоль для діагностики.

Такий підхід забезпечує плавну взаємодію: при натисканні на кнопку не відбувається перезавантаження, а кнопка одразу відображає новий стан улюбленої гри.

3.7.3 Карусель медіа-файлів на сторінці детальною інформації

У шаблоні game_detail реалізовано зручну Bootstrap-карусель яка відображає мультимедійний контент (Рисунок 11). Вона об’єднує в себе трейлер і скрішоти в єдиний інтерактивний блок який є дуже зручний для використання, тому що можливо продивитися усі цікаві аспекти гри а не шукати по інтернету якусь медіа-інформацію про гру(Лістинг 15).

Лістинг 15 — Реалізація каруселі

```
<!-- Carousel Content -->
<div class="carousel-inner ratio ratio-16x9 bg-dark">
  {% if game.trailer_url %}
    <div class="carousel-item active h-100">
      <iframe                                          src="{{
game.trailer_url|add:'?autoplay=0&mute=1' }}"
              title="{{ game.title }} Trailer"
              allowfullscreen
              class="w-100 h-100"
              allow="autoplay; encrypted-media"></iframe>
      <div class="carousel-caption d-none d-md-block bg-dark
bg-opacity-75 rounded-pill px-3 py-1">
        <p class="mb-0 small">{% trans "Game Trailer" %}</p>
      </div>
    </div>
  {% endif %}
```

```
  {% for shot in game.screenshots.all %}
    <div class="carousel-item {% if not game.trailer_url and
forloop.first %}active{% endif %} h-100">
      <img src="{{ shot.image.url }}"
           class="d-block w-100 h-100"
           alt="Screenshot {{ forloop.counter }}"
           style="object-fit: contain;">
      <div class="carousel-caption d-none d-md-block bg-dark
bg-opacity-75 rounded-pill px-3 py-1">
        <p class="mb-0 small">{% trans "Screenshot" %} {{
forloop.counter }}</p>
      </div>
    </div>
  {% endfor %}
</div>
```

— карусель задає пропорції 16:9 для медіавікна, щоб відео та зображення виглядали зручними для перегляду

Трейлер:

- перевіряє наявність посилання на відео;
- перший елемент позначається класом active, що робить його початковим слайдом.

Скріншоти:

- кожен запис із медіафайлів перетворюється на новий слайд;
- якщо трейлер відсутній, то перший скріншот отримує клас active;
- також відображається підпис скріншота.

Навігація:

- користувач може гортати трейлер та слайди за допомогою стрілок каруселі або чекати пока скриншоти прокручуються самостійно.

Допомагає лаконічно розмістити інформацію медіафайлів так щоб користувачу було зручніше використовувати продукт.

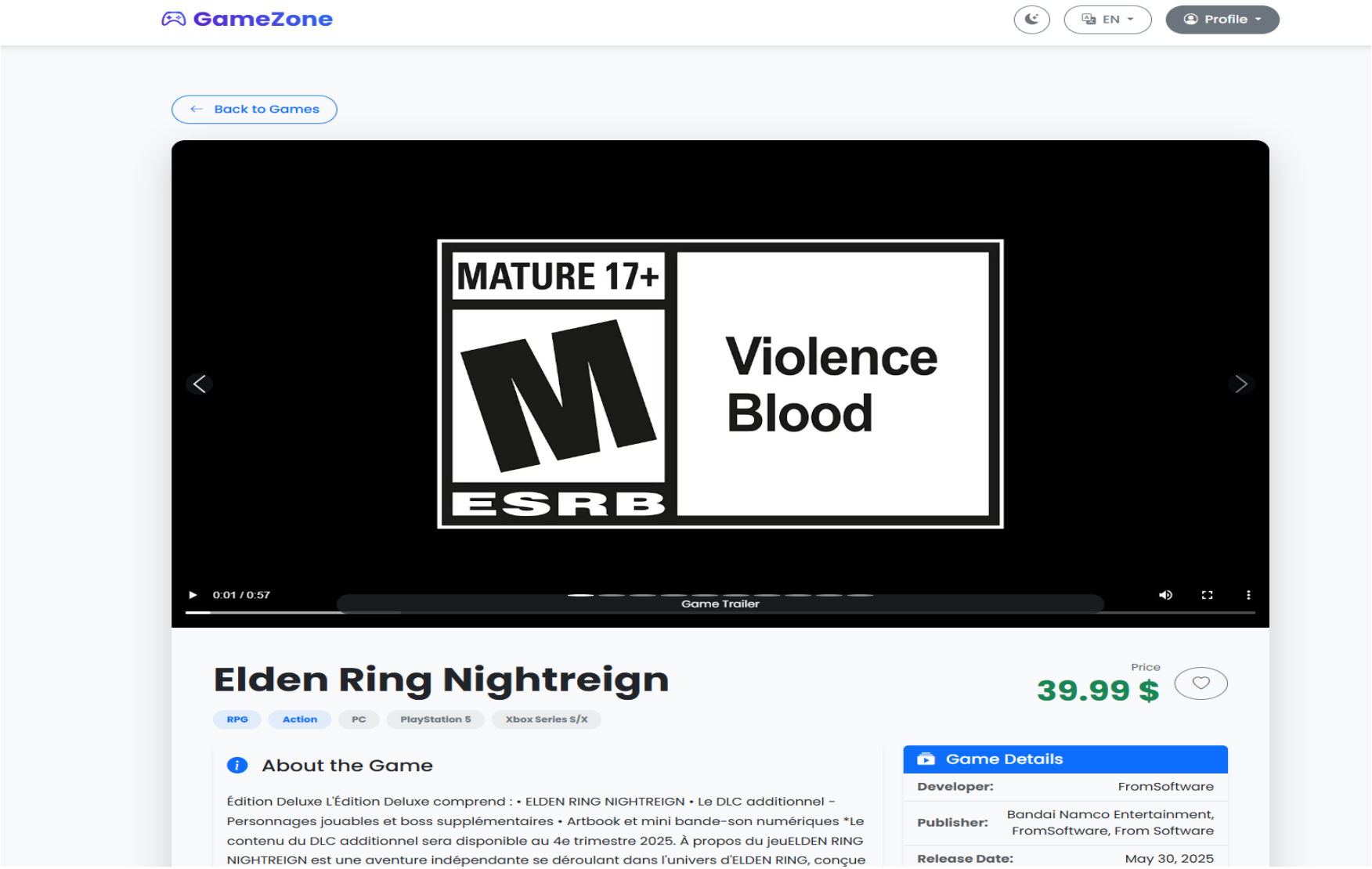


Рисунок 11 — *Відображає візуальну частину каруселі.*

3.8 Реєстрація, вхід та профіль користувача

Система аутентифікації на веб-застосунку надає персональні налаштування, гарантує безпечний доступ до функцій список «Улюблених ігор», виставлення рейтингу бо без акаунту застосунок буде просити вас ввійти у профіль або зареєструватись[5].

Реєстрація:

• Валідація полів перевірка унікальності нікнейму, відповідності паролів та мінімальної довжини.

• Метод *form.save* генерує запис у таблиці auth_user з автоматичним хешуванням паролю за алгоритмом.

• Після успішної реєстрації вас направляє на сторінку входу де ви можете одразу аутентифікуватися.

• Виведення повідомлення після створення акаунту (Лістинг 16).

Лістинг 16 — Виведення повідомлення

```
def register(request):
    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request,    "Обліковий    запис
створено успішно")
            return redirect('login')
    else:
        form = UserCreationForm()
    return   render(request,   'registration/register.html',
{'form': form})
```

Вхід у систему:

• Аунтефікація реалізована через стандартну форму;

• Перевірка облікових даних ім’я та пароль вони порівнюються з записами в БД;

• Функція логіну встановлює сесію та додає cookie

• Параметр next після входу надає можливість повернутись на початкову сторінку звідки було перенаправлення;

• Повідомлення про помилки виводяться у шаблоні якщо користувач невірно ввів логін чи пароль (Лістинг 17).

Лістинг 17 — Реалізація шаблону помилки

```
def login_view(request):
    form = AuthenticationForm(request, data=request.POST or
None)
    if form.is_valid():
        login(request, form.get_user())
        return      redirect(request.GET.get('next')      or
'profile')
    return    render(request,    'registration/login.html',
{'form': form})
```

Профіль користувача

Реалізований шаблон профілю (Лістинг 18) відображає персональні дані користувача, також надає інформацію о акаунті коли останній раз був залогінений чи нікнейм зі зміною аватару, також надається інформація про список «Улюблених ігор».[6]

Лістинг 18 — Реалізація шаблону профілю

```
{% extends "base.html" %}
{% block content %}
<h2>Профіль: {{ user.username }}</h2>
<ul class="list-group mb-3">
  <li   class="list-group-item"><strong>Email:</strong>   {{
user.email }}</li>
  <li                  class="list-group-item"><strong>Дата
реєстрації:</strong>  {{  user.date_joined|date:"d.m.Y  H:i"
}}</li>
</ul>

<h3>Улюблені ігри</h3>
<ul class="list-group mb-3">
  {% for fav in user.favoritegame_set.all %}
    <li class="list-group-item">
      <a  href="{%  url  'game_detail'  fav.game.id  %}">{{
fav.game.title }}</a>
```

```
      <span      class="text-muted      small">додано      {{
fav.added_at|date:"d.m.Y" }}</span>
    </li>{% empty %}
    <li class="list-group-item">Список порожній</li>
  {% endfor %}
</ul>
{% endblock %}
```

Цілі підсистеми

— в безпечному режимі тільки авторизовані користувачі можуть додавати в «Улюблені» та ставити оцінки це контролюється декораторами @login_required або перевіркою.

— інформація у профілі яка показує в інтерфейсі інформацію про акаунт додані у список «Улюблені ігри»

3.9 Функціональні та нефункціональні вимоги

Функціональні вимоги

1. Імпорт даних:
   - завантаження ігор та деталей з описом, жанром, платформою, системними вимогами, трейлером, скріншотами та цінами через API;
   - оновлення цін через окрему API.
2. Пошук і фільтрація:
   - пошук гри за введеною назвою;
   - відбір за жанрами, платформами, діапазоном цін та безкоштовними іграми;
   - комбіновані фільтри з динамічним підвантаженням.
3. Керування акаунтом:
   - реєстрація, вхід, вихід з сесії;
   - особистий профіль з переліком улюблених ігор та інформацією про акаунт.
4. Система «Улюблені ігри» та рейтингів:

- додавання та видалення ігор у «Улюблені ігри»;
- виставлення оцінки 1-5 із автоматичним перерахунком середнього рейтингу.

5. Адміністрування:
   - масове та ручне імпортування ігор;
   - адміністрування кожної з таблиці.
6. Інтерфейс користувача:
   - адаптивний дизайн із Bootstrap 5;
   - карусель медіаконтенту;
   - перемикач теми та мови й входом у профіль з використанням cookie.

Нефункціональні вимоги

1. Продуктивність:
   - час відповіді час відповіді при фільтрації та пагінації не перевищує 300 мс;
   - кешування результатів перекладу та складних ORM-запитів.
2. Масштабованість
   - можливість горизонтального розгортання Gunicorn-процесів за балансувальником Nginx;
   - відокремлені сервіси кешу та збереження медіа.
3. Надійність і відмовостійкість
   - повторні спроби HTTP-запитів до RAWG із backoff-стратегією;
   - автоматичні міграції БД та збір статичних файлів у процесі деплою;
   - централізоване логування помилок.
4. Безпека:
   - CSRF-захист, XSS-екранування, SQL-параметризація ORM;зберігання паролів із PBKDF2;
   - SSL-термінація на Nginx, HSTS.
5. Локалізація та доступність:
   - підтримка англійською та переклад елементів українською;

- зручна навігація, ARIA-атрибути для елементів форм.

6. Сумісність:

- коректне відображення в останніх версіях Chrome, Firefox та Edge;
- graceful degradation для старіших браузерів.

7. Підтримуваність:

- дотримання PEP8 і стандартів Django.

З виконанням цих вимог набувається необхідний рівень функціональності, якості обслуговування та зручності для кінцевого користувача та адміністратора системи [7].

3.10 Ручне тестування інтерфейсу

Щоб перевірити коректність роботи ключових функцій веб-застосунку були виконані ручні тести, які охоплювали навігацію, фільтрацію, автентифікацію та адміністративні дії. Таблиця тесту складається з кроків, очікуваного результату та фактичний результат (Таблиця 2).

Таблиця 2 — *Ручне тестування*

№	Назва тесту	Кроки тестування	Очікуваний результат	Фактичний результат	Статус
1	**2**	**3**	**4**	**5**	**6**
1	Перегляд каталогу ігор	1Відкрити головну сторінку. 2 Переконатися у наявності карток з іграми	Відображаються картки ігор з вірною інформацію	Відображення коректне, картка містить потрібну інформацію	успішно

Продовження таблиці 2 — *Ручне тестування*

1	2	3	4	5	6
2	Пошук за назвою	1 Ввести у поле «Пошук» назву «The witcher» 2Натиснути пошук	У списку залишаються лише ігри, назви яких містять «The witcher»	Пошук працює, результати відповідають запиту	Успішно
3	Фільтрація за жанром	1У випадаючому списку «Жанр» обрати «RPG» 2Натиснути «Фільтрувати»	Показуються тільки RPG-ігри	Фільтр спрацьовує, відображаються лише RPG	Успішно
4	Фільтрація за діапазоном цін	1Установити «Від 50$» та «До 100$» 2Натиснути «Фільтрувати»	У списку лише ігри з поточною ціною між 50 та 100 доларами	Відображаються ігри в заданому діапазоні	Успішно
5	Додавання в Улюблені	1Авторизуватися 2Перейти в деталі гри натиснути ♡ 3Перейти в профіль	Обрана гра з’являється в профілі в улюблених	У профіль гра додана відображається	успішно

Продовження таблиці 2 — *Ручне тестування*

1	2	3	4	5	6
6	Виставлення рейтингу	1Авторизуватися 2Перейти на сторінку гри 3Вибрати 1 із 5 у рейтингу	Після натискання оцінка гри зміниться на сторінці	Оцінка встановлена, середній рейтинг оновився	успішно
7	Реєстрація нового користувача	1Відкрити register 2Заповнити всі поля валідними даними 3Натиснути Зареєструватися	Перенаправлення на сторінку входу з повідомленням про успіх	Обліковий запис створено, повідомлення відображено	успішно
8	Авторизація	1Відкрити login 2Ввести коректні логін і пароль 3Натиснути «Увійти»	Переадресація на профіль, відображення імені користувача	Вхід успішний, профіль відкритий	успішно

Усі перевірки були виконані у Chrome. Ручне тестування підтвердило стабільність ключових сценаріїв, коректне відображення та взаємодію без помилок.

ВИСНОВКИ

1. Було проведено комплексний аналіз предметної області веб-застосунку для порівняння цін відеоігор, визначено ключові сутності: гри, платформи, жанри, ціни, рейтинги, скриншоти і їхні взаємозв’язки в реляційній БД.
2. Проведено порівняльний огляд існуючих рішень (IsThereAnyDeal, CheapShark, PriceCharting), виявлено їхні переваги й обмеження, що стало підставою для формулювання функціональних вимог до власного проєкту.
3. Розроблено логічну та фізичну архітектуру системи: реалізовано багаторівневу структуру “Presentation → Application → Data”, налаштовано моделі Django з проміжними таблицями для зв’язків many-to-many, організовано кешування Redis та зберігання медіафайлів на S3.
4. Реалізовано інтеграцію з зовнішніми API (RAWG, Steam) для автоматичного імпорту ігор, трейлерів, скриншотів та історії цін із забезпеченням повторних спроб і обробкою помилок.
5. Створено адаптивний фронтенд на базі Bootstrap 5 і JavaScript (ES6, AJAX): реалізовано динамічне фільтрування, карусель медіаконтенту, перемикач теми , функціонал «Улюблені» та рейтинги з миттєвим оновленням без перезавантаження сторінки.
6. В результаті виконання кваліфікаційної роботи створено готовий до розгортання веб-застосунок GameZone із повним циклом імпорту, зберігання та відображення даних про відеоігри, що відповідає встановленим функціональним та нефункціональним вимогам і може бути розширений новими джерелами даних і аналітичними модулями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Pearson Education, Limited, 2018. С. 20-25.
2. Shaw B., Badhwar S., Guest C., Chandra B. Web Development with Django, 2024. С. 45-60.
3. Freeman A., Robson E. Head First Design Patterns: A Brain-Friendly Guide. O'Reilly Media, Inc., 2021. С.72-90.
4. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. O'Reilly Media, Inc., 2013. С. 15-35.
5. Greenfeld A., Roy-Greenfeld D. Two Scoops of Django 3.x: Best Practices for Django Web Framework. Nomad Press, 2020. С. 110-125.
6. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O’Reilly Media, 2017. С. 200-215.
7. Laudon K. C., Laudon J. P. Management Information Systems: Managing the Digital Firm. 16th ed. Pearson, 2021. С. 240-255.
8. Baumgartner P., Malet Y. High Performance Django. O’Reilly Media, Inc., 2021. С. 60-80.
9. Belay A. D. Pro Docker for Python Developers. Apress, 2021. С. 90-110.
10. Vincent W. Django for Professionals: Production Websites with Python and Django. Independently published, 2020. С. 70-100.
11. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. Self-published, 2022. С. 40-90.
12. Garrison J., Nova K. Docker for Developers. O’Reilly Media, 2021.С.70-120
13. Percival H. J. W. Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript. 2nd ed., O’Reilly Media, 2021. С. 60-85.

14. RAWG. *RAWG Video Games Database API*.
URL: https://api.rawg.io/docs (дата звернення 03.05.2025).
15. ValveCorporation. *Steam Store*.
URL: https://store.steampowered.com/ (дата звернення30.04.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Тішков Владислав Віталійович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми “Програмне забезпечення систем”, адреса електронної пошти se121-22@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Розробка веб-сайту для пошуку та аналізу ігор за широким спектром характеристик**» 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 02.06.2025 _____ Тішков Владислав Віталійович
(підпис) (прізвище та ініціали) (студент)

Дата 02.06.2025 _____ Міхайлуца Олена Миколаївна
(підпис) (прізвище та ініціали) (керівник)