ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система для моніторингу динаміки цін для ігрових предметів на P2P маркетах на основі веб-скрейпінгу та API-інтеграцій»

на здобуття освітнього ступеня магістра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Роман ШУШВАЛ
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Роман ШУШВАЛ
група КНДМ-61

Керівник: Володимир ВАСИЛЕНКО
науковий ступінь, вчене звання к. т. н., доц.

Рецензент:
науковий ступінь, вчене звання
_____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Магістр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шушвалу Роману Вікторовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система для моніторингу динаміки цін для ігрових предметів на P2P маркетах на основі веб-скрейпінгу та API-інтеграцій»
_____

керівник кваліфікаційної роботи Володимир ВАСИЛЕНКО к. т. н., доц.
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024р. №320

2. Строк подання кваліфікаційної роботи «15» грудня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, сучасні методи статистичного аналізу, методи візуалізації даних, а також підходи до розробки програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Дослідження сучасних методів статистичного аналізу та візуалізації даних за допомогою спеціалізованого програмного забезпечення.

4.2. Проектування прототипу системи та вибір відповідних технологій для її реалізації.

4.3. Розробка та впровадження прототипу системи для візуалізації даних і проведення статистичного аналізу.

5. Перелік графічного матеріалу: *презентація*

5.1 Аналіз проблеми та огляд технологій.

5.2 Архітектура прототипу системи.

5.3 Демонстрація роботи прототипу.

5.4 Висновки та перспективи розвитку.

6. Дата видачі завдання«16» вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	16.09-27.09.24	виконано
2	Дослідження методів веб-скрейпінгу, API-інтеграції та їх застосування для аналізу даних.	30.09-11.10.24	виконано
3	Проектування прототипу системи та вибір технологій для її реалізації	14.10-25.10.24	виконано
4	Розробка серверної частини для збору та обробки даних з зовнішніх джерел.	28.10-08.11.24	виконано
5	Розробка клієнтської частини для візуалізації та роботи з даними.	11.11-22.12.24	виконано
6	Тестування головних компонентів створеного прототипу системи.	25.11-29.11.24	виконано
7	Оформлення роботи: вступ, висновки, реферат	02.12-06.12.24	виконано
8	Розробка демонстраційних матеріалів	09.12-13.12.24	виконано

Здобувач вищої освіти _____ Роман ШУШВАЛ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Володимир ВАСИЛЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 55 стор., 3 табл., 13 рис., 28 джерел.

Наукове завдання – розробка прототипу системи для моніторингу динаміки цін для ігрових предметів на P2P маркетах на основі веб-скрейпінгу та API-інтеграцій

Мета роботи – створення прототипу системи для моніторингу динаміки цін ігрових предметів на P2P маркетах.

Об'єкт дослідження – процес аналізу даних та їх візуалізації на основі динаміки цін ігрових предметів.

Предмет дослідження – інструменти та технології автоматизації аналізу та візуалізації даних, включаючи методи веб-скрейпінгу та API-інтеграцій.

Методи дослідження – аналіз і синтез сучасних технологій аналізу даних та їх візуалізації, алгоритмізація процесів обробки та графічного представлення інформації, застосування методів статистичного аналізу.

Короткий зміст роботи: У роботі проведено дослідження сучасних технологій автоматизації аналізу даних та їх графічного представлення. Розглянуто ефективні підходи до вдосконалення існуючих методів візуалізації та аналітики.

Розроблено прототип системи, який дозволяє автоматизувати процес моніторингу цін та аналізувати статистичні характеристики вибірок. Система інтегрує можливості читання даних із зовнішніх джерел, зокрема через API та веб-скрейпінг, та забезпечує ефективний інструмент для попереднього аналізу й візуалізації.

Ключові слова:МОНІТОРИНГ ЦІН, P2P МАРКЕТИ, ІГРОВІ ПРЕДМЕТИ, ВЕБ-СКРЕЙПІНГ, API-ІНТЕГРАЦІЯ, СТАТИСТИЧНИЙ АНАЛІЗ.

ABSTRACT

Text part of the master's qualification work: 60 pages, 13 pictures, 3 table, 28 sources.

Scientific task is development of a prototype system for monitoring the price dynamics of gaming items on P2P marketplaces based on web scraping and API integrations.

Goal of the work is to create a prototype system for monitoring the price dynamics of gaming items on P2P marketplaces.

Object of research – the process of data analysis and visualization based on the price dynamics of gaming items.

Subject of research – tools and technologies for automating data analysis and visualization, including web scraping and API integration methods.

Research methods – analysis and synthesis of modern technologies for data analysis and visualization, algorithmization of data processing and graphical representation processes, and application of statistical analysis methods.

Summary of the work: The study investigates modern technologies for automating data analysis and graphical representation. Effective approaches to improving existing visualization and analytics methods are examined.

A prototype system has been developed to automate the process of price monitoring and analyze the statistical characteristics of datasets. The system integrates the capability to retrieve data from external sources, including APIs and web scraping, and provides an efficient tool for preliminary analysis and visualization.

Keywords: price monitoring, P2P MARKETPLACES, GAMING ITEMS, WEB SCRAPING, API INTEGRATION, STATISTICAL ANALYSIS.

ЗМІСТ

ВСТУП

Сучасний ринок цифрових ігрових предметів на P2P (peer-to-peer) маркетплейсах демонструє високий рівень динамічності та непередбачуваності. Торгівля такими товарами, як скіни для зброї, унікальні артефакти чи віртуальні предмети обмеженого випуску, залежить від безлічі факторів: рівня попиту та пропозиції, сезонних тенденцій, впливу кіберспортивних подій, маркетингових кампаній та навіть змін у ігрових оновленнях. Така складна взаємодія чинників формує специфічний ринок, який значно відрізняється від традиційної електронної комерції.

Ігрові P2P маркетплейси, такі як Steam Market, DMarket, CS.MONEY та інші, стали майданчиками для купівлі-продажу ігрових предметів, обсяги яких вимірюються в мільйонах доларів щорічно. Водночас трейдери та гравці стикаються з низкою викликів: необхідністю оперативного аналізу величезних обсягів даних, відсутністю єдиного стандарту для структурованої інформації та недоступністю частини даних через політику платформ. У результаті ухвалення рішень щодо купівлі чи продажу ускладнюється, а ризики втрати коштів збільшуються.

Моніторинг цін на P2P маркетах має важливе значення, оскільки ціни на ігрові предмети формуються під впливом багатьох факторів, що швидко змінюються. Відсутність централізованого механізму обміну інформацією між платформами ускладнює доступ до даних, що, у свою чергу, впливає на ефективність ухвалення рішень трейдерами й інвесторами..

Для вирішення цих проблем використовується веб-скрейпінг і API-інтеграція, які дозволяють збирати, обробляти та аналізувати дані в автоматизованому режимі. Веб-скрейпінг дає змогу отримувати дані безпосередньо зі сторінок сайтів, де відображаються ціни, обсяги продажів та історія змін, навіть якщо офіційні API обмежені. API-інтеграція, зі свого боку,

забезпечує структурований доступ до надійних даних із платформ, знижуючи витрати на ресурси для збору інформації та мінімізуючи технічні ризики.

Об’єктами дослідження є:

Інтеграція серверної та клієнтської частини для забезпечення безперебійного обміну даними між базою даних і візуалізаційним інтерфейсом.

Реалізація функціональності моніторингу та збереження цінових даних із P2P платформ (Steam, DMarket, тощо) у реальному часі.

Автоматизація оновлення інформації за допомогою налаштування регулярних запитів до API і скрейпінгу для збору нових даних.

Оцінка ефективності системи через тестування її продуктивності, точності отриманих даних та зручності інтерфейсу.

Інтеграція із зовнішніми джерелами даних для забезпечення максимально широкого охоплення ринку ігрових предметів.

Ці аспекти дослідження дозволяють не лише створити систему для моніторингу, а й забезпечити додаткові інструменти для прийняття рішень на основі даних, що постійно оновлюються.

Візуалізація цінових трендів у реальному часі допомагає користувачам швидко і точно оцінити ситуацію на ринку. Інтерактивні інтерфейси, які демонструють варіації цін, історичні зміни та прогнозування, дозволяють трейдерам приймати обґрунтовані рішення щодо купівлі або продажу. Водночас, наявність регулярного оновлення інформації за допомогою API та веб-скрейпінгу дає можливість відстежувати ринкові зміни в межах декількох хвилин, що значно знижує ймовірність фінансових втрат та дає змогу швидко реагувати на зміни попиту і пропозиції.

1 ТЕОРЕТИЧНІ ОСНОВИ МОНІТОРИНГУ ЦІН НА P2P МАРКЕТАХ

1.1 Огляд P2P маркетплейсів для ігрових предметів

P2P (peer-to-peer) маркетплейси - це цифрові платформи, які забезпечують прямий обмін товарами, послугами або ресурсами між користувачами без необхідності залучення традиційного посередника. У контексті ігрових предметів, P2P маркетплейси дають змогу гравцям купувати, продавати чи обмінювати цифрові активи, такі як скіни для зброї, колекційні картки, унікальні артефакти або ігрову валюту.

Основні характеристики P2P маркетплейсів:

Прямий обмін між користувачами: P2P платформи забезпечують пряму взаємодію між покупцем і продавцем. Технологічна інфраструктура дозволяє автоматизувати основні етапи угоди: пошук товарів, укладання угоди, здійснення платежів і передачу активів.

Гарантії безпеки угод: Хоча обмін здійснюється між користувачами напряму, маркетплейс зазвичай виступає гарантом угоди, забезпечуючи прозорість і захист від шахрайства. Це може включати перевірку автентичності предметів, обробку транзакцій через платформу та забезпечення повернення коштів у разі спірних ситуацій.

Цифрові активи як основний товар: На P2P ринках для ігрових предметів основними товарами є цифрові об'єкти, що мають цінність у межах конкретних відеоігор. Їхня популярність визначається унікальністю, ігровою користю, естетичною привабливістю або рідкістю.

Відсутність централізованого посередника: Відмінною рисою P2P платформ є відсутність традиційного посередника. Проте платформа виконує роль цифрового хабу, який надає інструменти для здійснення угод і гарантує довіру між сторонами.

Гнучкість цінового формування: Ціни на P2P маркетплейсах визначаються попитом і пропозицією, що дозволяє користувачам самостійно встановлювати вартість своїх активів. Це створює динамічний ринок, де ціни постійно змінюються залежно від трендів і попиту.

Автоматизація транзакцій: Багато P2P платформ використовують смарт-контракти та інші інструменти для автоматизації транзакцій. Це знижує витрати часу на завершення угод і мінімізує ризик людської помилки.

Доступність для глобальної аудиторії: Завдяки цифровому формату, P2P маркетплейси охоплюють користувачів з усього світу. Це дозволяє гравцям торгувати активами незалежно від географічного розташування.

Комісії платформи: Платформи зазвичай стягують комісію з кожної угоди, що є джерелом їхнього доходу. Розмір комісії може варіюватися залежно від платформи.

P2P маркетплейси стають невід'ємною частиною цифрової економіки, сприяючи розвитку торгівлі ігровими предметами. Вони спрощують доступ до ринків, забезпечують зручність і розширюють можливості для трейдерів та гравців.

Найвідоміші P2P платформи :

1. **Steam Market**

Опис: Це один із найбільших маркетплейсів, інтегрований у платформу Steam, яка об’єднує мільйони гравців з усього світу [21]. Див рисунок 1.1.

Особливості: Пропонує великий вибір ігрових предметів із багатьох популярних ігор, включаючи CS:GO, Dota 2 та Team Fortress 2 та інші. Платформа дозволяє користувачам безпечно купувати та продавати предмети через внутрішній Steam-гаманець.

Недоліки: Основний обмежувальний фактор — неможливість виведення коштів із гаманця у реальні гроші. Комісія за транзакції становить до 15%. Також є обмеженняя по максимальній вартості предмету яка становить 2000$

Рисунок 1.1 – P2P маркетплейс Steam Market

2. DMarket

Опис: DMarket — це глобальна платформа для купівлі, продажу та обміну цифрових ігрових предметів [4]. Вона підтримує інтеграцію з популярними іграми, такими як Counter-Strike: Global Offensive (CS:GO), Dota 2 та Team Fortress 2, дозволяючи користувачам монетизувати свої ігрові активи. Платформа орієнтована на прозорість і безпеку, використовуючи блокчейн-технології для відстеження транзакцій. Див рисунок 1.2.

Особливості: DMarket забезпечує швидкі транзакції з низькими комісіями, що приваблює трейдерів та геймерів. Вона пропонує автоматизовані рішення для покупки та продажу, включаючи підтримку масових угод. Крім того, платформа дозволяє виводити кошти в реальних грошах, а також має зручний мобільний додаток, що робить її доступною для користувачів у будь-який час.

Недоліки: DMarket має низку недоліків, серед яких високі комісії за продаж, що знижують прибутковість для користувачів, обмежений функціонал для безкоштовних акаунтів, недостатня підтримка рідкісних чи нішевих предметів,

можливі затримки з виведенням коштів та залежність від ринкових умов, що може спричиняти нестабільність цін.

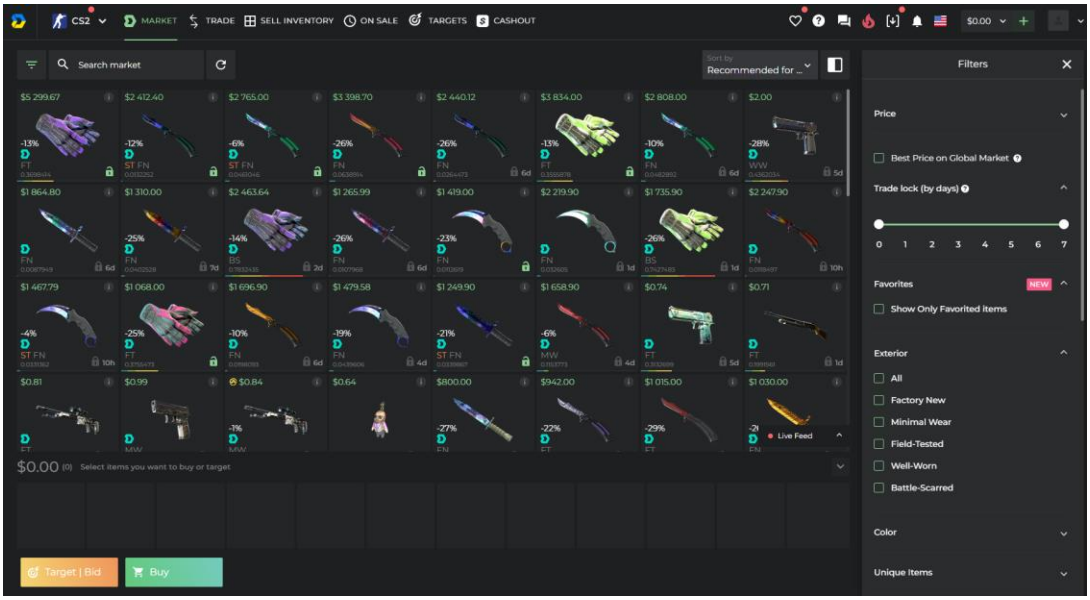


Рисунок 1.2 – P2P маркетплейс DMarket

3. CS.Money

Опис: CSMoney — це платформа для обміну, купівлі та продажу ігрових предметів, здебільшого для Counter-Strike: Global Offensive [3]. Вона надає користувачам можливість автоматизованих трейдів через ботів, забезпечуючи швидкі операції. CSMoney орієнтується як на геймерів, які шукають конкретні скіни для персоналізації, так і на трейдерів, які хочуть заробляти на перепродажі. Див рисунок 1.3.

Особливості: CSMoney пропонує зручний і інтуїтивний інтерфейс, що дозволяє користувачам легко знаходити потрібні предмети. Платформа підтримує функцію автоматичного підбору найкращих угод, а також надає інструменти для порівняння цін. CSMoney також пропонує великий каталог із популярними та рідкісними скінами, що робить її привабливою для гравців з різними запитами та бюджетами.

Недоліки: CSMoney має кілька суттєвих недоліків, які можуть впливати на досвід користувачів. По-перше, платформа встановлює комісію на транзакції, що

знижує прибутковість трейдингу, особливо при великих чи частих операціях. По-друге, сервіс не підтримує виведення коштів у реальні гроші, обмежуючи користувачів лише можливістю реінвестувати зароблені кошти в нові обміни чи покупки.

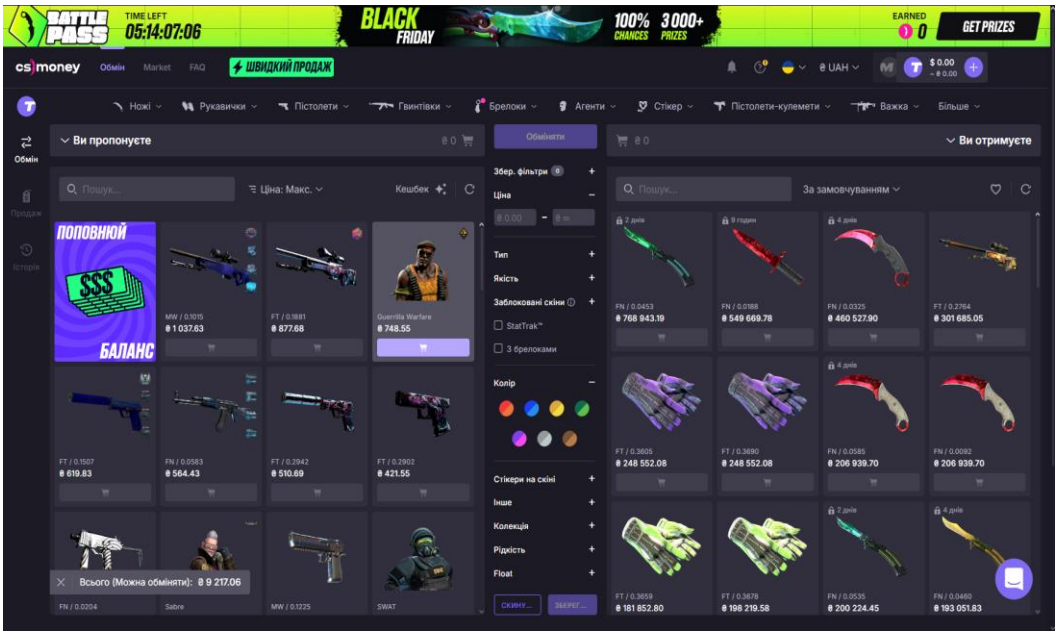


Рисунок 1.3 – P2P маркетплейс CS.Money

4. SkinBaron

Опис: SkinBaron — європейська P2P платформа, що спеціалізується на торгівлі скінами з ігор, таких як CS2 і Dota 2 [18]. Її ключова особливість — можливість продажу ігрових предметів із подальшим виведенням коштів у реальну валюту. Це робить платформу популярною серед трейдерів, які прагнуть не лише обмінюватися або покращувати свої скіни, а й монетизувати їх. Див рисунок 1.4.

Особливості: SkinBaron спеціалізується на торгівлі ігровими предметами з можливістю виведення реальних грошей, що робить його зручним для професійних трейдерів і користувачів, які хочуть монетизувати свої активи. Платформа пропонує зрозумілий інтерфейс для створення угод і забезпечує високий рівень безпеки транзакцій. Вона також доступна для міжнародної аудиторії, підтримуючи кілька валют для зручності користувачів.

Недоліки: SkinBaron також має свої обмеження. Головним недоліком є відносно висока комісія за транзакції, яка може зробити продаж скінів менш вигідним порівняно з іншими платформами. Крім того, доступ до деяких функцій, таких як виведення коштів, іноді супроводжується затримками через процедури верифікації. Інтерфейс платформи орієнтований переважно на досвідчених трейдерів, що може ускладнювати використання новачками. Ще одним мінусом є обмежений асортимент для певних ігор, оскільки SkinBaron зосереджується переважно на Counter-Strike 2.

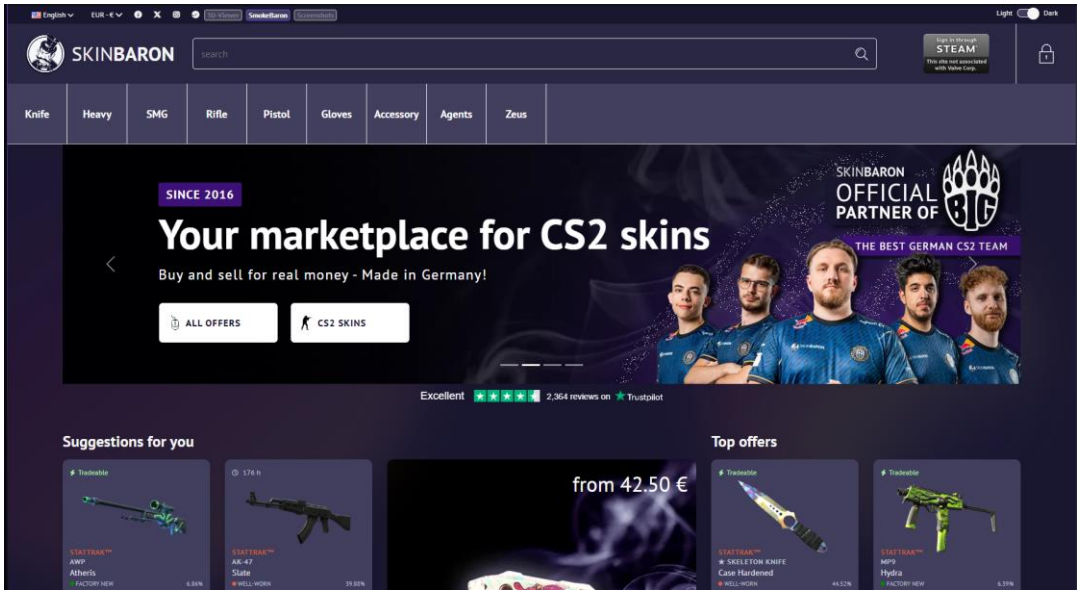


Рисунок 1.4 – P2P маркетплейс SkinBaron

5. Buff.163

Опис:Buff.163 — одна з найбільших P2P платформ для торгівлі ігровими предметами, яка набирає популярність в Азії, особливо в Китаї [2]. Платформа пропонує розширені можливості для аналізу ринку, забезпечуючи користувачів конкурентними цінами та додатковими інструментами для прийняття рішень. Див рисунок 1.5.

Особливості: Buff.163 орієнтований на азійський ринок і пропонує розширений функціонал для аналізу ринку ігрових предметів. Платформа надає графіки змін цін, аналітичні дані про популярність предметів і конкурентні ціни,

що робить її ідеальною для стратегічної торгівлі і завдяки системі відгуків Buff.163 забезпечує прозорість і довіру між користувачами.

Недоліки :Buff.163, попри свою популярність серед користувачів, має кілька недоліків. Найсуттєвішим є складність реєстрації та верифікації для міжнародних користувачів, оскільки платформа переважно орієнтована на китайський ринок. Крім того, процес виведення коштів з Buff.163 може бути повільним і пов'язаним із додатковими комісіями, що знижує привабливість для трейдерів. Інтерфейс також може бути неінтуїтивним для тих, хто не володіє китайською мовою, хоча й доступний автоматичний переклад. Обмеження щодо деяких регіонів та валют є ще одним недоліком, який ускладнює доступ трейдерам з інших країн.

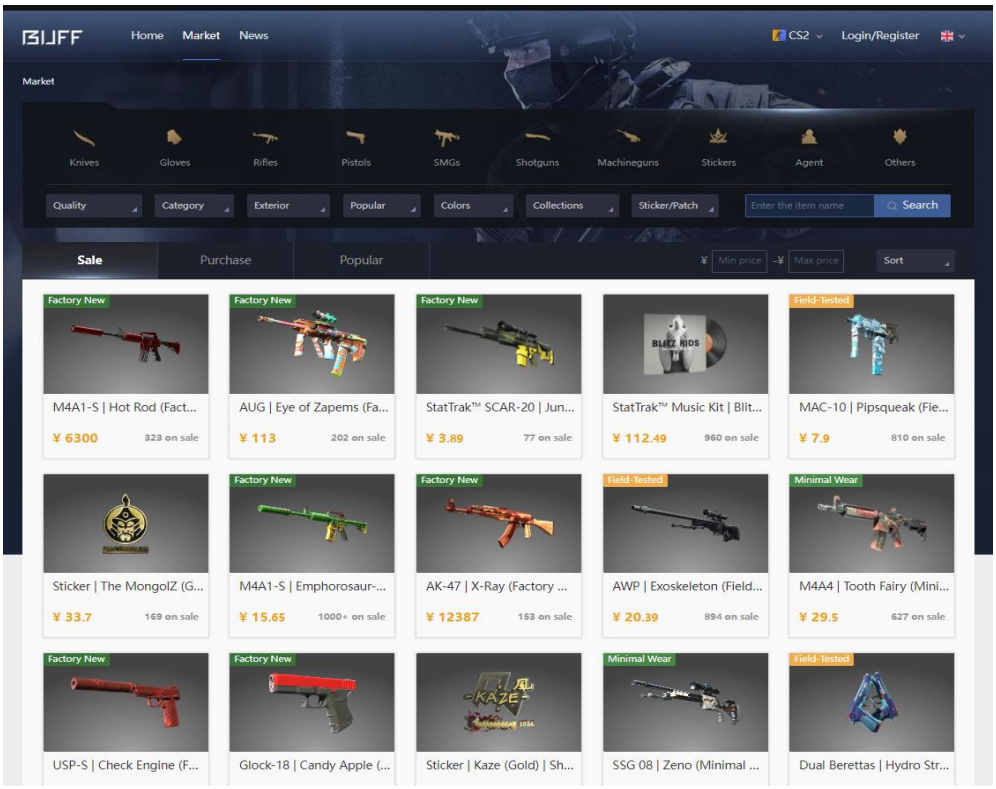


Рисунок 1.5 – P2P маркетплейс buff163

6. Avan market

Опис: Avan.market — це сучасна P2P платформа, що спеціалізується на торгівлі цифровими ігровими предметами, зокрема скінами для популярних ігор, таких як Counter-Strike: Global Offensive [1]. Сайт пропонує користувачам можливість купувати та продавати ігрові предмети за конкурентними цінами, забезпечуючи швидкі та безпечні транзакції. Див рисунок 1.6.

Особливості: Avan.market пропонує користувачам можливість моментального продажу скінів, що значно спрощує процес реалізації ігрових предметів. Завдяки цьому трейдери можуть швидко отримувати кошти без необхідності тривалого очікування на покупця. Платформа також забезпечує актуальні ринкові ціни, широкий асортимент предметів і зручний інтерфейс, який підходить як для новачків, так і для досвідчених користувачів. Інтеграція з популярними платіжними системами робить процес угод швидким і безпечним.

Недоліки: Avan.market, незважаючи на можливість швидкого продажу скінів, має свої недоліки. Основною проблемою є обмежений вибір доступних для купівлі та продажу предметів порівняно з великими платформами, такими як Steam чи DMarket. Комісії за продаж можуть бути вищими за середні на ринку, що знижує вигоду для трейдерів.

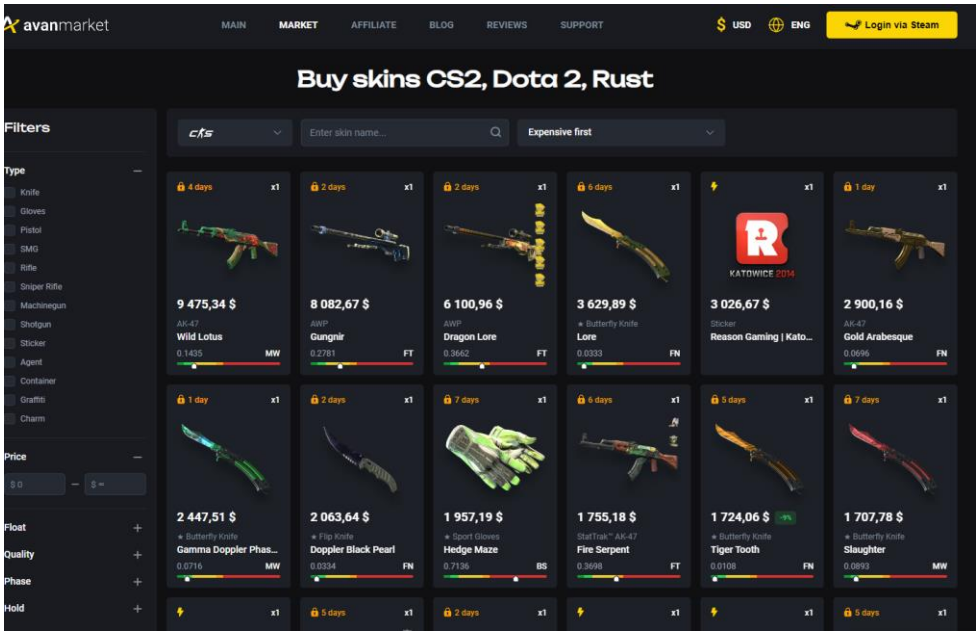


Рисунок 1.6 – P2P маркетплейс SkinBaron

1.2 Динаміка ціноутворення на P2P ринках: чинники впливу

Ціноутворення на P2P ринках є складним процесом, який залежить від багатьох факторів [11]. Ці платформи характеризуються постійними змінами вартості товарів, зокрема цифрових ігрових предметів, таких як скіни, колекційні картки, ігрова валюта або унікальні артефакти. Основні чинники, що впливають

на зміну цін, включають попит, рідкість, тренди серед користувачів, сезонність та оновлення у самих іграх.

Ціна на P2P маркетплейсах, як і на будь-якому іншому ринку, визначається співвідношенням попиту та пропозиції. Коли на певний ігровий предмет існує високий попит, ціна на нього зростає. Це може статися, наприклад, після великих оновлень гри або виходу нових сезонів, таких як у Counter-Strike 2 чи Dota 2, що призводить до збільшення вартості популярних скінів. З іншого боку, коли певний предмет стає менш актуальним або виходять новіші предмети, пропозиція перевищує попит, що спричиняє падіння цін.

Попит є основним чинником, який прямо впливає на вартість товару. Ігрові предмети, популярні серед гравців через їхній дизайн або прив’язаність до великих кіберспортивних подій, можуть значно зрости в ціні. Чим більше гравців зацікавлені в конкретному товарі, тим більший попит і, відповідно, вища ціна. Зміни в попиті можуть бути викликані маркетинговими кампаніями, новими релізами ігор чи оновленнями, які змінюють доступність чи популярність певних товарів.Рідкість є ще одним важливим фактором у формуванні цін. Чим менше одиниць певного предмета є у доступі, тим вища його цінність для користувачів. Рідкісні предмети часто є частиною обмежених серій, які не виробляються після завершення події або сезону. Наприклад, предмети, які випускаються в межах турнірів або спеціальних акцій, можуть мати значно вищу ціну завдяки обмеженій кількості.

Ціни на ігрові предмети також схильні до сезонних коливань. У періоди святкових розпродажів або під час кіберспортивних змагань, таких як турніри з Counter-Strike: Global Offensive, попит на певні предмети може зрости, що веде до тимчасового підвищення цін. Наприклад, під час літнього розпродажу Steam гравці часто інвестують більше в покупку віртуальних товарів, що підвищує їхню вартість.

Оновлення у самих іграх також суттєво впливають на ціни. Додавання нових предметів, змінення механік гри або інтерфейсу може призвести до зниження популярності старих предметів та, відповідно, до зниження їхньої

вартості. Навпаки, скасування випуску певного предмета чи його переробка може підвищити його цінність через рідкість і колекційну значимість. Кожне велике оновлення в грі змінює баланс ринку. Наприклад, додавання нових карт або зброї може призвести до зростання інтересу до певних предметів. У деяких випадках навіть незначні зміни в характеристиках ігрових елементів можуть змінити попит і вартість певних товарів.

Тренди серед гравців, зокрема популярність окремих кольорів, стилів або тематик, впливають на ціни. Наприклад, предмети, які використовують відомі стримери або професійні кіберспортсмени, часто стають об’єктами підвищеного попиту, що призводить до росту цін. Спільнота формує своєрідні "модні" тенденції, які змінюються з часом.

На ринку ігрових предметів нерідко зустрічаються спекуляції. Деякі професійні трейдери купують великі партії певного предмета з метою штучного створення дефіциту і підвищення його вартості. Це може бути вигідно для окремих трейдерів, але водночас створює ризики для звичайних гравців. Такі дії часто викликають короткострокову нестабільність, але ринок зазвичай відновлюється після того, як запаси повертаються в обіг.

На P2P маркетах, таких як DMarket чи SkinBaron, конкуренція між продавцями значно впливає на ціни. Якщо кілька користувачів виставляють однаковий предмет, вони прагнуть запропонувати конкурентну вартість, часто знижуючи ціну, щоб прискорити продаж. Це створює динамічну зміну ринку, коли ціни можуть коливатися в залежності від кількості доступних пропозицій. Ціна на один і той самий предмет може значно відрізнятися залежно від платформи. Наприклад, вартість предмета на Steam Market може бути значно вищою, ніж на Buff.163, через особливості платформи, такі як комісії, обмеження на виведення коштів, або відмінності в купівельній спроможності користувачів із різних регіонів.

На багатьох платформах існують комісії за продаж, які враховуються при формуванні ціни. Наприклад, на Steam Market продавці часто встановлюють ціни

з урахуванням комісії платформи, щоб отримати бажану суму. Це може створювати різницю між цінами на різних маркетах.

Поведінка гравців також є важливим фактором ціноутворення. Наприклад, гравці часто готові платити більше за предмети, які мають символічну або емоційну цінність (наприклад, скіни, пов'язані з улюбленими командами). У деяких випадках цінність предметів формується не лише їхньою рідкістю, але й асоціаціями або репутацією в ігровій спільноті.

Динаміка ціноутворення на P2P ринках є складним і багатогранним процесом. Вона залежить від взаємодії багатьох факторів, включаючи попит, рідкість, сезонні тренди та політику платформ. Розуміння цих процесів дозволяє трейдерам і гравцям адаптуватися до змін і приймати вигідніші рішення. Зрештою, ефективний аналіз ринку є ключем до успіху на таких платформах.

1.3 Огляд сучасних технологій збору даних

Веб-скрейпінг — це процес автоматичного збору даних з веб-сторінок за допомогою спеціальних програмних інструментів, що дозволяють отримувати структуровану інформацію з тексту, зображень, посилань та інших елементів веб-сторінки[28]. Зазвичай веб-скрейпінг використовується для збору великих обсягів даних, які необхідно аналізувати чи обробляти, таких як ціни, відгуки, або новини з різних сайтів. Скрейпінг часто здійснюється за допомогою бібліотек програмування, таких як BeautifulSoup (Python) або Scrapy, що дають змогу отримати HTML-код сторінки та витягти потрібні дані.

У контексті динамічних веб-сторінок, що завантажуються за допомогою JavaScript, веб-скрейпінг має деякі складнощі. Дані на таких сторінках часто не відображаються у початковому HTML-коді, а завантажуються додатково через запити до серверів або з баз даних. Для таких ситуацій використовуються додаткові інструменти, як Selenium або Puppeteer, які дозволяють взаємодіяти з веб-сторінками так, ніби це робить реальний користувач, що дає змогу отримувати і динамічно завантажені дані.

Веб-скрейпінг є ефективним інструментом для збору великих обсягів даних з інтернет-ресурсів, однак його застосування потребує уваги до етичних аспектів. Деякі вебсайти можуть обмежувати доступ до своїх даних через політики конфіденційності, тому важливо враховувати правила та умови користування платформами. Багато сайтів використовують технології, які блокують автоматичний доступ, наприклад, CAPTCHA або обмеження на частоту запитів. Тому для здійснення веб-скрейпінгу може знадобитися використання проксі-серверів або спеціальних методів обходу цих захистів, що додає складності в процесі збору інформації.

Незважаючи на технічні перешкоди, веб-скрейпінг все одно залишається популярним методом для збору цінових даних, новин, відгуків та іншої інформації з інтернет-ресурсів, особливо в умовах P2P платформ. Для отримання даних з таких платформ, як Steam або DMarket, важливо правильно налаштувати інструменти збору, щоб отримувати актуальну інформацію в режимі реального часу. Інтерфейси API таких платформ, хоча й дозволяють доступ до певної частини даних, можуть бути обмежені в порівнянні з повним обсягом даних, доступних через скрейпінг. Тому для комплексного аналізу даних на P2P маркетах часто використовують комбінацію цих технологій.

API (Application Programming Interface) є важливим інструментом для збору даних, що надається безпосередньо платформами для інтеграції з іншими додатками або системами. API дозволяє автоматизувати процес збору та обміну даними між різними програмними середовищами, що робить його дуже корисним у моніторингу цін на P2P маркетах. На відміну від веб-скрейпінгу, який потребує парсингу HTML-коду сторінок, API забезпечує більш структурований доступ до даних у вигляді JSON, XML або інших форматів, що значно полегшує їх обробку та аналіз.

API надають можливість здійснювати запити на отримання певної інформації, наприклад, поточних цін на ігрові предмети, історії продажів, а також статистичних даних. Зазвичай ці платформи обмежують кількість запитів до API за певний проміжок часу, що дозволяє запобігти перевантаженню серверів і

зберегти ефективність їх роботи. Відповідно до умов користування API, користувачам можуть бути надані різні рівні доступу в залежності від їхнього статусу, що може бути корисно для трейдерів та розробників додатків, які потребують регулярного оновлення даних для аналізу цінових трендів.

Однією з переваг використання API є стабільність та надійність у порівнянні з веб-скрейпінгом. За допомогою API можна отримувати доступ до актуальних даних без необхідності зайвого навантаження на сервери чи використання обхідних шляхів для відсутніх можливостей. Більше того, API дозволяють здійснювати запити, орієнтуючись на конкретні параметри або фільтри, що дає можливість отримати лише релевантні дані, зменшуючи час і ресурси, необхідні для обробки інформації.

Проте, незважаючи на свої переваги, API має і певні обмеження. В першу чергу, доступ до API може бути платним або обмеженим для звичайних користувачів. Багато платформ надають доступ лише до частини своїх даних через API або мають обмеження на кількість запитів, що можуть бути зроблені за день. Крім того, зміни в політиці API або оновлення самих платформ можуть призвести до зміни способів доступу або навіть повної втрати доступу до даних. Це вимагає від розробників постійного моніторингу та адаптації своїх інструментів збору даних до нових умов.

API також є більш етично прийнятним варіантом для збору даних, оскільки платформи надають такі інтерфейси з метою полегшення інтеграції та взаємодії з іншими сервісами. Це дає змогу більш прозоро та легально отримувати дані, без порушення умов використання сайту чи необхідності обходити технічні обмеження, які можуть бути присутніми на рівні HTML-сторінок. Тому для розробників, які прагнуть створювати стабільні й ефективні системи збору даних, API є одним із найкращих варіантів.

Веб-скрейпінг та API є двома основними методами збору даних в сучасних цифрових системах. Кожен із цих підходів має свої особливості, переваги та обмеження, які визначають їх ефективність в різних умовах:

Ефективність і точність. Веб-скрейпінг є потужним інструментом для збору даних із різноманітних веб-ресурсів, зокрема тих, що не надають API доступу. Однак цей метод може бути менш точним, оскільки змінюються структури веб-сторінок або контент на сторінках, що може вимагати оновлення скриптів для збору даних. У порівнянні з API, який надає чітко структуровані дані, веб-скрейпінг може призвести до помилок або неповних даних, якщо структура веб-сторінки змінюється. API дозволяє працювати з даними, що надаються в стандартизованому форматі (наприклад, JSON або XML), що забезпечує високу точність і стабільність.

Легальність і етичні аспекти. З юридичної точки зору, API надає законний доступ до даних, оскільки цей доступ часто регулюється умовами користування API і вимогами конфіденційності. Веб-скрейпінг, з іншого боку, може порушувати правила використання веб-сайтів, особливо якщо збір даних здійснюється без дозволу. Деякі компанії навіть використовують технічні обмеження (наприклад, блокування IP-адрес) для обмеження автоматичного збору даних. Етичні питання також можуть виникати, якщо скрейпінг здійснюється без належного дотримання прав користувачів і ресурсів, що використовуються.

Потреба в технічних ресурсах. Веб-скрейпінг вимагає значних технічних ресурсів, зокрема для розробки і підтримки інструментів, які здатні коректно витягувати дані з різних сторінок. Це може включати налаштування алгоритмів для обробки складних структур веб-сторінок або зміни в структурі HTML. У той же час, API значно спрощує процес збору даних, оскільки для цього зазвичай потрібно лише здійснити запит і обробити результат. Однак API може мати обмеження на кількість запитів або необхідність обробки великої кількості даних, що також вимагає певних ресурсів для оптимізації.

Гнучкість і масштабованість. Веб-скрейпінг може бути більш гнучким, оскільки дозволяє збирати дані з будь-яких веб-сайтів, навіть тих, що не мають публічно доступного API. Це особливо важливо для збору даних з платформ, де API можуть бути обмежені або відсутні. Однак такий підхід може бути важчим у масштабуванні, оскільки для кожного сайту можуть бути потрібні окремі

налаштування. У той же час, API забезпечує зручність і стабільність, особливо для роботи з великими обсягами даних на регулярній основі. Більшість великих платформ надають добре документовані API, що значно спрощує роботу зі збором даних, але знову ж таки, обмеження API можуть ставати проблемою при великих обсягах запитів.

Безпека. Веб-скрейпінг може бути більш уразливим до атак або помилок, оскільки збір даних здійснюється безпосередньо з веб-сторінок. Окрім цього, відсутність контролю за доступом може призвести до збору конфіденційної або чутливої інформації. У разі використання API без належної автентифікації або обмежень доступу це також може викликати серйозні проблеми. Однак API дозволяє вбудувати надійні механізми безпеки та аутентифікації, що знижує ризики витоку даних і зловживань.

Таким чином, вибір між веб-скрейпінгом і API залежить від конкретних потреб проекту, доступних ресурсів і важливості точності даних. API є більш стабільним і надійним рішенням для збору структурованої інформації, в той час як веб-скрейпінг дає більшу гнучкість, але з певними ризиками і технічними труднощами.

1.4 Висновки з аналізу теоретичних основ

P2P маркетплейси стали важливими елементами цифрової економіки, оскільки вони створюють нові можливості для обміну товарами та послугами без посередників [11]. Ці платформи, такі як eBay, Airbnb або Steam, дозволяють приватним особам і малим підприємствам брати участь у глобальних ринках, що раніше було недоступно. Вони сприяють розвитку економіки спільного використання та економії часу і ресурсів, що має величезне значення для підвищення ефективності ринкових відносин. Крім того, P2P платформи знижують витрати для споживачів, надаючи їм можливість вибору серед безлічі варіантів і встановлювати власні ціни на товари та послуги. Це дозволяє досягти більшої прозорості ринку та забезпечує рівні можливості для всіх учасників.

Завдяки своїй дистрибуції та доступу до нових цифрових інструментів, P2P маркетплейси також сприяють розвитку підприємництва, даючи змогу малим бізнесам знизити витрати на посередників і мати прямий контакт зі споживачами. Вони мають потенціал не лише для підтримки стартапів і нових підприємств, а й для розвитку нових бізнес-моделей, орієнтованих на спільне споживання. Таким чином, P2P ринки стали рушійною силою для інновацій в економіці, сприяючи зростанню малого та середнього бізнесу, а також розширенню можливостей для споживачів.

Моніторинг цін є важливим інструментом для користувачів та трейдерів на P2P маркетплейсах, оскільки дозволяє відстежувати коливання цін на конкретні товари або послуги в режимі реального часу. Це дає можливість оцінювати найвигідніші моменти для покупки чи продажу, а також прогнозувати зміни на ринку. Для трейдерів, це особливо важливо, оскільки завдяки точному моніторингу цін вони можуть адаптувати свої стратегії і максимізувати прибуток, орієнтуючись на актуальні тренди та попит. Завдяки моніторингу можна швидко реагувати на зміни ринкових умов і ухвалювати обґрунтовані рішення.

Для користувачів моніторинг цін дає можливість отримати кращу пропозицію, порівнюючи різні варіанти на ринку і вибираючи найкращу ціну відповідно до своїх потреб. Також, ці інструменти допомагають уникнути переплат, надаючи доступ до найбільш конкурентоспроможних пропозицій на ринку. Моніторинг цін стає незамінним інструментом для тих, хто активно займається торгівлею чи інвестує в цифрові товари, такі як ігрові предмети, оскільки дозволяє мінімізувати ризики і збільшити прибутковість угод.

Перспективи розвитку технологій збору даних для аналізу P2P ринків є досить великими, оскільки з кожним роком зростає кількість і складність даних, доступних для обробки. Веб-скрейпінг і API інтеграції дозволяють автоматизувати процес збору і аналізу даних, що є важливим аспектом для ефективного моніторингу P2P ринків. Веб-скрейпінг, зокрема, забезпечує доступ до інформації, яка може бути недоступна через традиційні API, оскільки дозволяє витягувати дані прямо з веб-сторінок. Це особливо корисно на платформах, де

відкритий доступ до даних обмежений, або де API не можуть забезпечити всього необхідного обсягу інформації. З розвитком технологій штучного інтелекту, машинного навчання та великих даних, процеси збору і обробки даних на P2P платформах стають ще більш ефективними та точними.

Водночас, є потреба в удосконаленні інструментів для збору даних у реальному часі, що дозволить трейдерам і користувачам приймати швидкі рішення. Однією з таких технологій є використання інтелектуальних алгоритмів для прогнозування змін цін на основі великої кількості зібраних даних. Це дозволить забезпечити більш точні прогнози та зменшити ризики, пов’язані з торговими угодами. Також, із впровадженням більш потужних аналітичних інструментів з’являється можливість для глибшого аналізу тенденцій на ринку, що дозволить краще прогнозувати попит і пропозицію та оптимізувати стратегії продажу.

Зростання популярності технологій збору та аналізу даних для P2P ринків відкриває нові можливості для більш гнучких та адаптивних торгових систем. Інтеграція технологій для автоматичного збору даних із різних джерел, таких як соціальні медіа, форуми та спеціалізовані платформи, також створює нові можливості для отримання точнішої інформації про ринок. У майбутньому можна очікувати, що ці технології будуть активно вдосконалюватися, що дозволить ще більш точно відображати реальні зміни на ринку і підвищити ефективність процесів торгівлі на P2P платформах.

2 АНАЛІЗ ПРОГРАМНО-ТЕХНІЧНИХ РІШЕНЬ СТВОРЕННЯ МОНІТОРИНГУ P2P МАРКЕТІВ СКІНІВ

Для досягнення цих цілей важливо ретельно підходити до вибору архітектури системи. Вона визначає, наскільки ефективно система зможе обробляти великі обсяги даних, адаптуватися до зростання навантаження та забезпечувати надійну роботу. Різні архітектурні підходи мають свої переваги й недоліки, що дозволяє підібрати оптимальне рішення відповідно до специфіки проєкту. У наступних підрозділах будуть проаналізовані найпоширеніші архітектури систем, зокрема монолітна, дворівнева, трирівнева та мікросервісна, для створення систем моніторингу P2P маркетів скінів.

2.1 Архітектури

Архітектури — це концептуальні підходи до організації програмних систем, які визначають спосіб взаємодії різних компонентів і рівнів системи. Вони дозволяють забезпечити ефективність, масштабованість і надійність роботи додатків. Вибір архітектури залежить від вимог до системи, зокрема її продуктивності, здатності до адаптації та стійкості до збоїв. У цьому розділі будуть розглянуті основні архітектурні моделі, включаючи монолітну, дворівневу, трирівневу та мікросервісну архітектури, їхні переваги, недоліки та приклади використання.

2.2.1 Монолітна архітектура

Монолітна архітектура є традиційним підходом до побудови програмних систем, де вся функціональність додатка реалізована в єдиному кодовому базисі та запускається як один процес. Усі компоненти, включаючи інтерфейс користувача, бізнес-логіку та управління даними, інтегровані в одну структуру.

Переваги монолітної архітектури:

Простота розробки: Завдяки цілісному підходу команда розробників може швидко почати роботу над проєктом без необхідності інтеграції окремих компонентів.

Швидке розгортання: Увесь додаток збирається в єдине розгортання, що знижує складність у налаштуванні інфраструктури.

Мінімальні затрати на інфраструктуру: Для роботи додатку часто достатньо одного серверу чи хостингу.

Недоліки монолітної архітектури:

Складність внесення змін: У великих системах зміна одного модуля може впливати на весь додаток, що підвищує ризик виникнення помилок.

Обмеження масштабування: Система масштабується лише в цілому, що може призвести до надмірного використання ресурсів для компонентів, які не потребують високої продуктивності.

Труднощі у впровадженні нових технологій: Через єдність архітектури перехід на нові технології або інструменти вимагає значних зусиль.

Коли доцільно використовувати монолітну архітектуру:

У невеликих проєктах, де важлива швидкість розробки та розгортання.

У проєктах з низьким рівнем складності, де система не потребує значного масштабу або гнучкості.

Для стартапів, які прагнуть мінімізувати витрати на розробку та інфраструктуру.

Монолітна архітектура залишається популярним вибором для багатьох програмних систем, особливо на етапі початкової розробки. Однак із зростанням проєкту та його складності може виникнути необхідність у переході до інших архітектурних підходів, таких як мікросервіси чи багаторівневі моделі.

2.1.2 Дворівнева архітектура

Дворівнева архітектура є одним із базових підходів до побудови програмних систем, у якому функціональність поділяється на два основні рівні: клієнтський

(frontend) і серверний (backend). Цей підхід використовується для спрощення організації додатку, розділяючи завдання обробки даних і взаємодії з користувачем.

Двошарова архітектура включає клієнтський і серверний рівні. Клієнтський рівень відповідає за інтерфейс користувача та обробку введених даних, а також може представлений у вигляді веб-додатку, мобільного додатку або настільного клієнта. Серверний рівень забезпечує бізнес-логіку, обробку запитів клієнта та взаємодію з базами даних. Крім того, він може здійснювати управління правами доступу, автентифікацію користувачів та інші функції, що забезпечують безпеку та ефективність системи.

Переваги дворівневої архітектури:

1. Простота розробки та підтримки: Завдяки чіткому розділенню логіки клієнтської та серверної частин полегшується робота з кодовою базою.
2. Покращена продуктивність: Значна частина обчислень переноситься на сервер, що знижує навантаження на клієнтський пристрій.
3. Гнучкість клієнтського інтерфейсу: Клієнт може бути реалізований у різних технологіях (наприклад, веб-додаток, мобільний додаток).

Недоліки дворівневої архітектури:

1. Можливе перевантаження серверу: У випадку значного збільшення кількості клієнтів сервер може стати вузьким місцем.
2. Обмежена масштабованість: У більшості реалізацій дворівневої архітектури складно масштабувати серверну частину незалежно від клієнтської.
3. Залежність клієнт-сервер: Пряма взаємодія між клієнтом і сервером може обмежувати можливості оптимізації роботи з даними.

2.1.3 Трирівнева архітектура

Трирівнева архітектура (Three-tier architecture) — це тип архітектури програмного забезпечення, яка ділить додаток на три основні рівні або шари для

виконання різних функцій. Кожен рівень відповідає певній частині процесу обробки даних і взаємодії користувача з системою.

Трирівнева архітектура складається з трьох основних рівнів: верхнього, середнього та нижнього. Верхній рівень (Presentation Layer) відповідає за інтерфейс користувача, забезпечуючи взаємодію між користувачем і програмою, як-от веб-інтерфейс або мобільний додаток. Середній рівень (Logic Layer) обробляє бізнес-логіку та взаємодіє з базою даних або іншими зовнішніми ресурсами для обробки запитів. Нижній рівень (Data Layer) керує збереженням і доступом до даних, забезпечуючи безпечний доступ до інформації. Ця архітектура дозволяє чітко розділити функції та підвищити масштабованість та підтримку додатків.

Переваги трирівневої архітектури:

Модульність — Кожен рівень можна легко змінювати та масштабувати окремо без впливу на інші рівні.

Чітке розмежування ролей — Відокремлення інтерфейсу користувача, бізнес-логіки та зберігання даних дозволяє покращити структурованість додатка.

Краща підтримка та обслуговування — Розділення рівнів сприяє простішому обслуговуванню, оскільки зміни на одному рівні не впливають на інші.

Масштабованість — Трирівнева архітектура дозволяє легке масштабування кожного рівня окремо, що робить систему гнучкою під потреби користувачів.

Безпека — Розділення даних на рівні дозволяє краще керувати доступом до інформації та зменшує ризики безпеки.

Ізоляція бізнес-логіки — Відокремлення бізнес-логіки від інтерфейсу дозволяє легше створювати, модифікувати та тестувати додаток.

Краща продуктивність — Завдяки чіткому розділенню рівнів, знижується ризик конфліктів та підвищується ефективність обробки запитів.

Трирівнева архітектура має такі недоліки: складність інтеграції між рівнями, що потребує спеціальних інтерфейсів; високі витрати на розробку та

обслуговування через додатковий рівень логіки; затримки у виконанні через багаторівневу обробку даних; проблеми з масштабованістю та внесенням змін, якщо інфраструктура недостатньо спланована; ризики безпеки під час передачі даних між рівнями; обмежену гнучкість, коли зміни в одному рівні можуть впливати на інші; та значні витрати на підтримку через складність системи. Ці недоліки вимагають ретельного планування та оптимізації під час реалізації архітектури.

2.1.4 Мікросервісна архітектура

Мікросервісна архітектура — це сучасний підхід до розробки програмних систем, який передбачає поділ додатка на невеликі, незалежні сервіси, що виконують окремі функції та взаємодіють між собою через стандартизовані інтерфейси, такі як API. Кожен мікросервіс є самостійним і може бути розгорнутий, масштабований та оновлений незалежно від інших.

Мікросервісна архітектура є підходом до побудови програмних систем, у якому система розділена на набір незалежних сервісів, кожен із яких відповідає за виконання конкретної бізнес-функції. Це один із ключових принципів сучасної розробки, що дозволяє адаптувати системи до змін і забезпечувати їхню високу продуктивність та стійкість.

Основні принципи мікросервісної архітектури передбачають поділ системи на окремі сервіси, кожен із яких реалізує чітко визначену функціональність, наприклад, керування користувачами, обробку транзакцій чи роботу з інвентаризацією, працюючи незалежно, що спрощує обслуговування та впровадження змін. Ці сервіси є автономними модулями, які можуть використовувати різні технології, мови програмування чи бази даних, забезпечуючи гнучкість у розробці. Мікросервіси взаємодіють між собою через стандартизовані протоколи, такі як HTTP/REST або gRPC, чи системи обміну повідомленнями, наприклад Kafka або RabbitMQ. Така модульність дозволяє

масштабувати кожен сервіс окремо відповідно до навантаження, що оптимізує витрати на ресурси та підвищує ефективність системи.

Архітектурні особливості мікросервісної архітектури полягають у використанні інтерфейсів взаємодії через API, що забезпечує стандартизовану інтеграцію сервісів за допомогою протоколів HTTP(S), WebSocket, AMQP або брокерів повідомлень, таких як Kafka чи RabbitMQ. Мікросервіси зазвичай розгортаються на різних серверах або контейнерах, що підвищує відмовостійкість системи. Для управління розгортанням і масштабуванням часто застосовуються платформи контейнеризації, такі як Docker, та оркестраційні системи на кшталт Kubernetes. Крім того, кожен мікросервіс може мати власну базу даних, яка найкраще відповідає його потребам, наприклад, SQL для фінансових операцій чи NoSQL для обробки журналів подій, що додає гнучкості та ефективності в роботі.

Мікросервісна архітектура має низку переваг, серед яких гнучкість розробки, що дозволяє різним командам працювати над окремими сервісами паралельно, прискорюючи впровадження нових функцій. Завдяки локалізації змін модифікації в одному сервісі не впливають на інші, що мінімізує ризики помилок. Відмовостійкість системи забезпечується тим, що збій одного сервісу не порушує роботу решти системи за умови правильної архітектури. Продуктивність і масштабованість досягаються через можливість масштабувати лише високонавантажені сервіси, що дозволяє ефективно використовувати ресурси. Також мікросервіси дають змогу застосовувати різні технології: сервіси можуть бути створені на різних мовах програмування або з використанням оптимальних баз даних залежно від завдань.

Мікросервісна архітектура має і певні недоліки, серед яких складність у проєктуванні, оскільки потрібно ретельно продумувати взаємодію між сервісами та стандарти API. Значними є витрати на інфраструктуру, адже для ефективної роботи потрібні інструменти для контейнеризації (наприклад, Docker), оркестрації (Kubernetes) та моніторингу. Розподілена природа мікросервісів створює труднощі у комунікації, оскільки потребує складних механізмів маршрутизації, обробки збоїв і ведення логів. Крім того, моніторинг і логування стають викликом,

оскільки необхідно відстежувати стан кожного окремого сервісу, що ускладнює управління системою загалом.

Інструменти для мікросервісної архітектури забезпечують ефективність і стабільність системи. Для контейнеризації використовується Docker, що дозволяє ізолювати сервіси, спрощуючи їх розгортання. Оркестрацію сервісів здійснює Kubernetes, який керує розгортанням, масштабуванням і забезпечує відмовостійкість. Для комунікації між сервісами застосовуються системи обміну повідомленнями, такі як Apache Kafka або RabbitMQ, що забезпечують надійний обмін даними. Моніторинг системи виконується за допомогою Prometheus та Grafana, які дозволяють відстежувати продуктивність і стан сервісів у реальному часі. Для логування широко використовується ELK Stack (Elasticsearch, Logstash, Kibana), який забезпечує збір, аналіз і візуалізацію логів для ефективного управління.

Мікросервісну архітектуру доцільно використовувати у великих і високонавантажених проєктах, де ключовими є масштабованість і відмовостійкість. Вона підходить для складних систем із численними функціональними модулями, які потребують чіткого поділу логіки. Крім того, цей підхід ефективний у проєктах, що вимагають частого розгортання оновлень, швидкої адаптації до змін і мінімізації ризиків при внесенні модифікацій.

Мікросервісна архітектура ідеально підходить для створення сучасних розподілених систем, які повинні бути гнучкими, масштабованими та надійними. Проте вона вимагає відповідних ресурсів та досвіду команди для впровадження.

2.2 Основні технології

2.2.1 Мови програмування

Для створення систем моніторингу P2P маркетів скінів використовуються різні мови програмування, кожна з яких має свої особливості та переваги залежно від завдань:

Python — це високорівнева мова програмування, яка широко використовується завдяки своїй простоті, читабельності та універсальності. Вона надає розробникам можливість швидко створювати якісний код для різних задач, що робить її популярним вибором для проєктів у різних галузях.

Ця мова вирізняється своєю простотою та читабельністю, оскільки синтаксис мови наближений до природної, що полегшує її вивчення й прискорює розробку. Завдяки багатій екосистемі бібліотек і фреймворків, таких як Django і Flask для веб-розробки, Pandas і NumPy для обробки даних, а також TensorFlow і PyTorch для машинного навчання, Python забезпечує широкий спектр можливостей. Його кросплатформеність дозволяє запускати програми на різних операційних системах без змін у коді, а швидкість розробки значно підвищується завдяки простому синтаксису та готовим інструментам.

Python є універсальною мовою програмування, яка підходить як для невеликих проєктів, так і для розробки масштабних систем. Величезна спільнота розробників активно створює нові інструменти й підтримує існуючі, забезпечуючи широкий вибір готових рішень. Завдяки низькому порогу входу, Python дозволяє швидко розпочати розробку, що робить його ідеальним вибором для проєктів із короткими термінами реалізації.

Однак Python має і певні обмеження. Він поступається компільованим мовам, таким як Java чи C++, у швидкодії, що може бути критичним для систем із високими вимогами до продуктивності. Крім того, через механізм Global Interpreter Lock (GIL) Python може демонструвати низьку ефективність у сценаріях, де необхідна багатоядерна обробка.

Python широко застосовується у системах моніторингу завдяки своїй гнучкості та багатому набору інструментів. Для обробки даних використовуються бібліотеки, як-от Pandas і Scikit-learn, які дозволяють ефективно працювати з великими обсягами даних, проводити аналіз і будувати прогнози. Мова чудово підходить для автоматизації процесів збору даних із різних джерел, наприклад, через API або веб-скрапінг із використанням бібліотек BeautifulSoup і Scrapy. Python також забезпечує швидку розробку бекенду за допомогою фреймворків

Flask і FastAPI, що дозволяє створювати REST API для обміну даними між мікросервісами. Крім того, Python легко інтегрується з базами даних, такими як PostgreSQL або MongoDB, та системами обміну повідомленнями, зокрема RabbitMQ і Kafka, забезпечуючи ефективну взаємодію між компонентами системи.

Java — це об’єктно-орієнтована мова програмування, яка відома своєю продуктивністю, стабільністю та універсальністю. Завдяки своїм можливостям вона широко застосовується для розробки масштабних корпоративних систем, мобільних додатків, вебсервісів і розподілених систем. Java залишається одним із найпопулярніших виборів для розробників завдяки своїм особливостям і багатій екосистемі.

Основна перевага Java полягає в її кросплатформеності: програми компілюються в байт-код, який виконується на Java Virtual Machine (JVM), що дозволяє запускати їх на будь-якій платформі без змін у коді. Java підтримує об’єктно-орієнтовану парадигму програмування, що забезпечує модульність, повторне використання коду та легку підтримку великих проєктів. Її продуктивність підвищується завдяки JIT-компіляції, яка оптимізує виконання програм під час роботи. Мова має багату екосистему фреймворків, таких як Spring і Hibernate для розробки бекенду, та інструментів для роботи з великими даними, наприклад, Apache Hadoop і Spark.

Java є ідеальною для розробки масштабованих і високонадійних систем. Її багатий набір бібліотек і активна спільнота розробників забезпечують широкий спектр готових рішень для різних завдань. Завдяки підтримці багатопоточності Java добре підходить для високонавантажених систем, а її стабільність робить її основним вибором для довгострокових корпоративних проєктів. Принцип "Write Once, Run Anywhere" значно спрощує процес розробки та розгортання.

Водночас Java має певні обмеження. Її синтаксис складніший у порівнянні з мовами на кшталт Python, що може збільшувати час розробки, особливо для початківців. Додатково, JVM споживає більше ресурсів, що може бути важливим чинником для невеликих проєктів або систем із обмеженими можливостями

обладнання. Попри підтримку багатопоточності, реалізація конкурентних задач іноді вимагає значних зусиль через складність синхронізації потоків.

Java широко застосовується у системах моніторингу завдяки своїм можливостям і стабільності. Для обробки великих обсягів даних Java інтегрується з платформами Apache Kafka, Hadoop і Spark, що дозволяє аналізувати дані та будувати прогнози. Завдяки своїй багатопоточності та розподіленій природі Java є основою для створення високонадійних бекенд-систем. Фреймворки Spring Boot і Quarkus дозволяють швидко розробляти REST API для комунікації між мікросервісами. Java також забезпечує ефективну взаємодію з базами даних, такими як MySQL, PostgreSQL чи MongoDB, а також із системами обміну повідомленнями, зокрема RabbitMQ і Kafka, що дозволяє створювати масштабовані та надійні рішення для моніторингу.

Next.js — це сучасний фреймворк для розробки вебзастосунків на основі React, який вирізняється своєю продуктивністю, гнучкістю та зручністю використання. Завдяки вбудованим інструментам і функціям Next.js активно застосовується для створення динамічних і масштабованих вебпроєктів, зокрема корпоративних платформ, інтернет-магазинів і контент-орієнтованих сайтів.

Однією з ключових особливостей Next.js є серверний рендеринг (Server-Side Rendering, SSR) і статична генерація (Static Site Generation, SSG), які забезпечують швидке завантаження сторінок і покращення SEO. Завдяки підтримці API-роутів Next.js дозволяє створювати бекенд-логіку в межах того ж самого застосунку, спрощуючи інфраструктуру. Вбудовані можливості для роботи з маршрутизацією, автоматичною оптимізацією зображень і підключенням CSS/SCSS роблять розробку швидкою та зручною. Також фреймворк забезпечує інтеграцію з популярними інструментами, такими як Tailwind CSS, TypeScript і різноманітними CMS.

Next.js підходить як для створення невеликих вебсайтів, так і для масштабних вебзастосунків із високими вимогами до продуктивності. Завдяки своїй універсальності, фреймворк дозволяє легко впроваджувати сучасні підходи до розробки, такі як Jamstack, що робить його чудовим вибором для проєктів,

орієнтованих на високу швидкість і доступність. Підтримка гібридних підходів до рендерингу (SSR і SSG в одному застосунку) дозволяє ефективно адаптуватися до змінних потреб проєкту.

Втім, Next.js має і свої обмеження. Наприклад, для великих і складних застосунків може знадобитися додаткове налаштування серверної інфраструктури, а розробникам може бути потрібний досвід роботи з React. Крім того, для деяких функцій, таких як інтеграція з нестандартними бекенд-системами, може знадобитися більше часу та зусиль, ніж у традиційних серверних фреймворках.

Next.js широко використовується у системах моніторингу завдяки своїй здатності швидко створювати зручні для користувачів інтерфейси. Наприклад, фреймворк дозволяє ефективно відображати аналітичні дані, що оновлюються в реальному часі, завдяки інтеграції з API-роутами та бібліотеками для роботи з вебсокетами. Для побудови інтерфейсів Next.js забезпечує інтеграцію з бібліотеками візуалізації, такими як Chart.js і D3.js. Фреймворк також дозволяє створювати динамічні панелі моніторингу, використовуючи можливості SSR для швидкого завантаження даних і інтеграцію з бекенд-системами через REST або GraphQL.

2.2.2 Docker та kubernetes

Docker та Kubernetes - це комбінація технологій, яка забезпечує ефективну контейнеризацію та оркестрацію додатків, що широко використовується у сучасній розробці та розгортанні програмного забезпечення. Вона забезпечує масштабованість, портативність і автоматизацію процесів, що робить її ідеальним вибором для мікросервісної архітектури.

Docker — це платформа для контейнеризації, яка дозволяє розробникам створювати, розгортати та запускати програми в ізольованих середовищах, званих контейнерами. Контейнери включають весь необхідний код, залежності та

середовище виконання для запуску програм, що робить їх портативними та передбачуваними незалежно від платформи.

Docker надає можливості ізоляції, забезпечуючи кожному контейнеру власні ресурси (CPU, пам’ять, мережу), що дозволяє запускати кілька додатків на одному сервері без конфліктів. Контейнери є портативними й можуть працювати на будь-якій платформі, яка підтримує Docker, незалежно від операційної системи. Швидкість розгортання значно перевищує віртуальні машини завдяки легкій архітектурі контейнерів. Масштабована екосистема, зокрема Docker Hub, надає доступ до великої кількості готових образів, спрощуючи розробку та розгортання.

Типовими сценаріями використання Docker є контейнеризація мікросервісної архітектури, де кожен мікросервіс упакований в окремий контейнер, що забезпечує модульність і масштабованість. У тестуванні Docker дозволяє створювати однакові середовища незалежно від локальних налаштувань розробників. У CI/CD Docker інтегрується в конвеєри безперервної інтеграції та доставки, полегшуючи збірку, тестування та розгортання додатків. У хмарних сервісах Docker використовується для створення портативних додатків, які можна запускати у хмарі без змін у конфігурації.

Docker є основою сучасних технологій розробки та розгортання програм. У поєднанні з такими інструментами, як Kubernetes, він дозволяє створювати масштабовані, модульні й ефективні системи, забезпечуючи високу продуктивність і швидкість розробки.

Kubernetes - це система для оркестрації контейнерів, яка автоматизує розгортання, масштабування та управління контейнеризованими додатками. Вона дозволяє ефективно керувати кластерами серверів і контейнерів, забезпечуючи надійність і високу доступність.

Kubernetes - це потужна система оркестрації контейнерів, яка керує групами контейнерів (подами), розподіляючи їх на різних вузлах кластера для балансування навантаження та забезпечення високої доступності. Вона автоматично масштабує контейнери залежно від навантаження, забезпечуючи ефективне використання ресурсів, і підтримує відновлення після збоїв,

перезапускаючи контейнери або пересуваючи їх на інші вузли. Конфігурації додатків описуються у вигляді YAML/JSON файлів, що дозволяє легко зберігати, версіонувати й повторно використовувати налаштування. Kubernetes автоматично відкриває контейнери один для одного, розподіляє трафік між ними для рівномірного навантаження й забезпечує безпечну взаємодію між сервісами завдяки налаштуванню мережевих політик. Система підтримує безперебійні оновлення додатків (ролинг-апдейти) із можливістю реверту, що дозволяє уникнути простоїв і збоїв під час переходу на нові версії.

Kubernetes широко використовується для управління мікросервісною архітектурою, забезпечуючи масштабованість і високу доступність розподілених сервісів. Його інтеграція в конвеєри CI/CD дозволяє автоматизувати розгортання додатків і керування версіями. Система підтримує гібридну хмару та мультихмарність, забезпечуючи запуск додатків як у локальному середовищі, так і в різних хмарних провайдерів. Для високонавантажених систем Kubernetes ефективно масштабує сервіси, забезпечуючи їхню стабільну роботу під час пікових навантажень.

Завдяки своєму гнучкому підходу до управління контейнерами, Kubernetes є ідеальним рішенням для сучасних IT-інфраструктур, де швидкість та надійність є ключовими факторами. Його можливості автоматизації, зокрема моніторинг, логування, обробка подій і безпека, дозволяють розробникам та адміністраторам систем ефективно підтримувати та оптимізувати роботу додатків. Крім того, зростання спільноти користувачів і постійне оновлення екосистеми Kubernetes гарантують, що система залишається актуальною і продовжує задовольняти найсучасніші потреби в оркестрації контейнерів.

2.2.3 Apache Kafka

Apache Kafka — це децентралізована платформа для обробки потокових даних, яка використовується для збирання, зберігання та передавання подій в реальному часі. Основна концепція Kafka базується на топіках (topics), кожен з

яких представляє логічний потік даних. Кожен топік складається з послідовності подій, які можуть бути підписані різними споживачами, що дозволяє масштабувати обробку даних між різними сервісами або компонентами.

Kafka призначена для роботи з великими обсягами даних і має високу продуктивність завдяки своєму дистрибутивному підходу. Дані можуть бути збережені у вигляді журналу, що дозволяє довгострокове зберігання інформації і можливість обробки історичних подій. Kafka також підтримує горизонтальне масштабування, роблячи її ідеальною для систем, що потребують обробки великої кількості запитів в реальному часі.

Kafka працює на основі топіків та журналів, що дозволяє зберігати події в часовій послідовності. Кожен топік представляє логічний потік даних, а журнали зберігають історію подій, що забезпечує масштабованість і можливість обробки потокових даних у реальному часі. Дані записуються до Kafka у вигляді подій, які можуть бути спожиті різними підписниками залежно від їхніх потреб. Це дозволяє створювати гнучкі системи обробки даних з підтримкою масштабування на кілька вузлів.

На відміну від Kafka, RabbitMQ використовує класичну модель "відправник-отримувач" з чергами (queues). Кожне повідомлення знаходиться в черзі, яка служить для передачі даних між різними компонентами. Таким чином, RabbitMQ більше орієнтована на процес передачі повідомлень між двома сторонами, що робить її менш гнучкою для обробки великих потоків даних.

Kafka чудово підходить для обробки великих потоків даних, забезпечуючи високу продуктивність завдяки дистрибутивній архітектурі та підтримці горизонтального масштабування. Вона може обробляти сотні тисяч запитів за секунду, що робить її ідеальним рішенням для систем реального часу та потокової обробки. Kafka також підтримує збереження даних у журналах для довгострокового зберігання, що підвищує її ефективність у задачах обробки великих даних.

RabbitMQ також забезпечує високу продуктивність, але більше підходить для середніх обсягів даних. Її продуктивність може бути обмежена для великих

потоків даних через використання класичної моделі черг, що не забезпечує горизонтального масштабування з такою ефективністю, як у Kafka.

Kafka має вбудовану підтримку горизонтального масштабування, що дозволяє додавати нові вузли без втрати продуктивності. Це робить її ідеальною для обробки великих обсягів даних та забезпечує високий рівень доступності даних навіть за великих навантажень. Завдяки децентралізованій архітектурі Kafka може легко підтримувати додавання вузлів для масштабування системи.

На відміну від неї, RabbitMQ також підтримує масштабування, однак його здатність обробляти величезні обсяги даних обмежена через використання черг. Це обмеження пов'язане з архітектурними особливостями, які не дозволяють RabbitMQ розподіляти дані між вузлами так само ефективно, як Kafka. Kafka зберігає дані у вигляді журналів (logs), що забезпечує довгострокове зберігання історії подій. Це дозволяє підтримку відновлення даних у разі збоїв, а також гарантує, що події можна переглядати та обробляти протягом тривалого часу.

RabbitMQ зберігає повідомлення лише на рівні черг, що призводить до тимчасового зберігання повідомлень. Це обмежує можливості RabbitMQ для довгострокового зберігання даних, і після обробки повідомлення можуть бути видалені. Таким чином, RabbitMQ не підходить для задач, які потребують збереження історичних даних або підтримки довгострокових архівів.

Таким чином, Kafka чудово підходить для масштабованих рішень із обробкою потоків даних, з можливістю зберігання великих обсягів історичних подій, тоді як RabbitMQ добре працює для інтеграції між сервісами та обміну невеликими обсягами повідомлень у більш простих сценаріях.

2.3 Бази данних

2.3.1 SQL бази данних

SQL (Structured Query Language) — це мова запитів до реляційних баз даних, яка використовується для управління та маніпуляції даними. Реляційні

бази даних організовують дані у вигляді таблиць, які складаються з рядків і стовпців, а зв’язки між ними визначаються за допомогою ключів. Основні концепції SQL включають створення таблиць, внесення змін, вилучення та запити до баз даних.

Дані в SQL базах даних зберігаються у формі реляційних таблиць. Кожна таблиця складається зі стовпців (колонок), що представляють атрибути даних, та рядків (рядків), які представляють конкретні записи або об’єкти. Це дозволяє впорядкувати дані та забезпечує легкість доступу до інформації.

SQL дозволяє виконувати різні операції з даними, включаючи:

SELECT – отримання даних з таблиць.

INSERT – додавання нових записів у таблиці.

UPDATE – оновлення існуючих записів.

DELETE – видалення записів з таблиць.

SQL підтримує різні типи даних, які дозволяють налаштовувати таблиці відповідно до специфічних вимог. Серед основних типів даних є текстові (VARCHAR, CHAR), числові (INTEGER, FLOAT), дати (DATE, TIMESTAMP), булеві (BOOLEAN) та інші, що дозволяють зберігати різноманітні дані відповідно до конкретних завдань.

SQL дозволяє встановлювати різні типи зв’язків між таблицями, такі як "один до одного", "один до багатьох" та "багато до багатьох". Ці зв’язки допомагають організувати та підтримувати цілісність даних у базі. Крім того, використання індексів підвищує ефективність виконання запитів, що забезпечує швидкий пошук інформації у великих таблицях.

Один до одного (1:1) один запис у першій таблиці відповідає одному запису в іншій таблиці. Використовується, коли інформація логічно розподілена між двома таблицями, але кожен запис є унікальним. Приклад таблиця users та таблиця user_profiles, де кожен користувач має лише один профіль.

Один до багатьох (1:N) один запис у першій таблиці може бути пов'язаний із багатьма записами в іншій таблиці. Це найпоширеніший тип зв'язків у базах

даних. Приклад таблиця categories і таблиця items, де кожна категорія містить багато предметів.

Багато до одного (N:1) це зворотній тип зв’язку "один до багатьох". Багато записів з однієї таблиці пов'язані з одним записом в іншій таблиці. Приклад багато предметів у таблиці items можуть належати одній категорії в таблиці categories.

Багато до багатьох (M:N) один запис у першій таблиці може бути пов'язаний із багатьма записами в іншій таблиці, і навпаки. Для реалізації цього зв'язку потрібна проміжна таблиця (наприклад, item_markets), яка містить зовнішні ключі до обох таблиць.Приклад предмети (items) можуть продаватися на багатьох платформах (markets), і кожна платформа може містити багато предметів.На рис 2.1 зображено приклад одного з зв’язків SQL баз данних

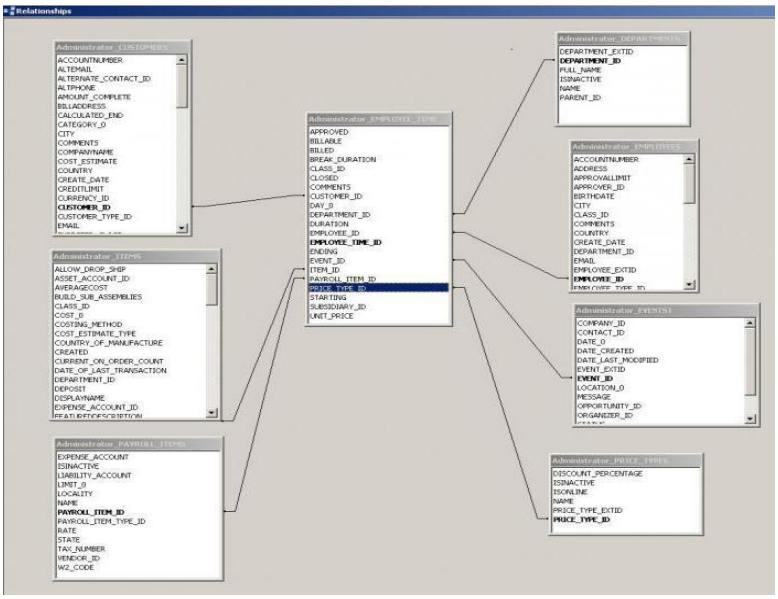


Рисунок 2.1 – Приклад одного з зв’язків SQL баз данних

SQL бази даних підтримують транзакції, що дозволяє забезпечити цілісність даних навіть у разі виконання множинних взаємопов’язаних операцій. Завдяки цьому можна гарантувати, що всі дії в рамках транзакції будуть виконані або повністю, або не виконані взагалі, якщо виникне помилка. Для цього використовуються різні рівні ізоляції, такі як Read Uncommitted, Read Committed, Repeatable Read та Serializable, які визначають, як транзакції взаємодіють одна з одною.

Крім того, SQL бази даних здатні обробляти великі обсяги даних завдяки підтримці оптимізації запитів, реплікації, резервного копіювання та відновлення.

Ці функції забезпечують високу продуктивність, надійність і доступність системи, навіть у масштабованих середовищах із високими вимогами до зберігання та обробки даних.

Популярні SQL бази даних включають PostgreSQL, яка є відкритою реляційною СУБД з високим рівнем масштабованості та підтримкою численних розширень, і MySQL, одну з найпопулярніших баз даних для веб-застосунків, що забезпечує високу продуктивність. Oracle Database виступає як корпоративне рішення з розширеними можливостями для управління даними, а Microsoft SQL Server, розроблений Microsoft, відзначається відмінною інтеграцією з їхніми продуктами, такими як .NET і Azure.

Таким чином, SQL бази даних мають широке застосування у таких сферах, як фінанси, електронна комерція, аналітика даних, управління запасами тощо.

2.3.2 NoSQL бази данних

NoSQL бази даних — це нереляційні системи управління базами даних, які забезпечують гнучкість і масштабованість, що особливо важливо для сучасних великих даних та додатків у реальному часі. Вони побудовані з акцентом на простоту розширення та обробку різнорідних даних, які не підходять під строгі структури SQL баз.

NoSQL бази даних характеризуються відсутністю реляційної моделі, оскільки для зберігання даних вони не використовують таблиці. Натомість дані організовуються в документах, графах, наборах ключів-значень або колонках, залежно від типу бази. Вони забезпечують гнучкість схеми, дозволяючи зберігати дані без визначення фіксованої структури, що зручно для роботи з динамічними даними, які змінюються з часом.

Завдяки горизонтальному масштабуванню NoSQL бази можуть легко додавати нові сервери у кластер для обробки зростаючих навантажень. Їхня проста архітектура забезпечує високу швидкість читання і запису, що робить їх ідеальними для додатків із великим обсягом даних у реальному часі. Крім того,

вони дозволяють зберігати структуровані, напівструктуровані або неструктуровані дані, забезпечуючи динамічне зберігання для різних типів інформації.

Типи NoSQL баз даних поділяються на кілька категорій залежно від способу зберігання та організації даних. Документні бази даних зберігають дані у вигляді документів (JSON, BSON, XML), що робить їх зручними для додатків зі складною структурою даних. Прикладами таких баз є MongoDB та CouchDB. Бази даних типу ключ-значення організовують інформацію у вигляді пар ключ-значення, що підходить для простих запитів, кешування та обробки даних із мінімальними взаємозв'язками. До популярних прикладів належать Redis та DynamoDB. Колонкові бази даних зберігають дані у вигляді стовпців, а не рядків, що забезпечує швидку обробку великих обсягів інформації, особливо в аналітичних системах. Відомі приклади — Apache Cassandra та HBase. Графові бази даних використовують вузли та ребра для представлення зв'язків між об'єктами, що робить їх ідеальними для соціальних мереж, рекомендаційних систем і аналізу мереж. Прикладами графових баз є Neo4j та Amazon Neptune.

NoSQL бази даних характеризуються відсутністю реляційної моделі, оскільки вони не використовують таблиці для зберігання даних. Дані можуть бути організовані у вигляді документів, графів, наборів ключів-значень або колонок, залежно від типу бази. Завдяки гнучкій схемі NoSQL бази дозволяють зберігати дані без фіксованої структури, що є зручним для роботи з динамічними та змінними даними. Вони підтримують горизонтальне масштабування, дозволяючи додавати нові сервери для обробки зростаючих навантажень. Їхня проста архітектура забезпечує високу швидкість читання та запису, що робить їх ідеальними для застосунків з великим обсягом даних у реальному часі. Крім того, NoSQL бази даних дозволяють зберігати структуровані, напівструктуровані та неструктуровані дані, забезпечуючи динамічне зберігання для різних типів інформації. На таблиці 2.1 зображено порівняння SQL та NoSQL баз даних

Таблиця 2.1 – Порівняння SQL та NoSQL баз данних

Характеристика	SQL	NoSQL
Структура даних	Таблиці з фіксованою схемою	Динамічна, документи, графи, ключ-значення
Запити	Стандартна мова SQL	Специфічні API або запити
Масштабованість	Вертикальна	Горизонтальна
Тип даних	Структуровані	Напівструктуровані або неструктуровані
Використання	Фінансові системи, ERP	Великі дані, реальний час, мобільні додатки

Одним з найпопулярніших noSQL баз данних є MongoDB — це документно-орієнтована NoSQL база даних, що зберігає дані у вигляді документів у форматі JSON або BSON (бінарна версія JSON). Завдяки своїй гнучкій схемі MongoDB дозволяє зберігати документи з різною структурою, що робить її ідеальною для додатків, де дані часто змінюються або мають складну структуру.

MongoDB підтримує горизонтальне масштабування через механізм шардінгу, який дозволяє розподіляти дані між кількома серверами для обробки великих обсягів даних і високих навантажень. Це робить базу даних ефективною для роботи з великими розподіленими системами.

Операції у MongoDB виконуються швидко завдяки індексуванню, яке полегшує пошук необхідних даних, та підтримці кешування для підвищення продуктивності. Для забезпечення надійності MongoDB використовує реплікацію, що дозволяє автоматично створювати копії даних на різних серверах для відновлення у випадку збоїв.

Основною особливістю MongoDB є вбудована агрегація, яка дозволяє виконувати складні аналітичні запити, обробляючи та трансформуючи дані на рівні бази. Крім того, MongoDB добре інтегрується з різними мовами

програмування, такими як Python, JavaScript, Java та інші, що робить її зручною для розробників.

MongoDB широко використовується в системах реального часу, великих веб-застосунках, IoT-платформах та аналітичних рішеннях завдяки своїй продуктивності, масштабованості та гнучкості.

3 РОЗРОБКА ПРОЕКТУ

3.1 Інженерія вимог системи для моніторингу динаміки цін для ігрових предметів

На підставі проведеного аналізу програмно-технічних рішень для створення систем моніторингу динаміки цін на ігрові предмети на P2P-маркетах сформовано основне завдання поставленої роботи:

спроектувати та розробити веб-застосунок для збору, аналізу та відображення змін цін на ігрові предмети, ґрунтуючись на веб-скрейпінгу та API-інтеграціях. Для створення веб-застосунку обраний підхід клієнт-серверної архітектури, яка враховує сучасні тенденції та забезпечує масштабованість системи.

Основними вимогами до проекту є:

1. Забезпечення захисту даних від несанкціонованого доступу та конфіденційність інформації користувачів.
2. Зменшення затримки завантаження та оновлення даних.
3. Створення функціональної моделі для аналізу динаміки цін.
4. Розробка серверної частини з інтеграцією API основних P2P-платформ.
5. Реалізація системи для збору даних за допомогою веб-скрейпінгу.
6. Розробка клієнтської частини, що забезпечує інтуїтивно зрозумілий інтерфейс і функціонал.

Для розробки клієнтської частини було використано фреймворк Next.js. Вибір Next.js обґрунтовано його характеристиками, які відповідають вимогам системи моніторингу динаміки цін, зокрема можливістю створення ефективних SSR (Server-Side Rendering) і SPA (Single Page Application) рішень.

Серед ключових переваг Next.js для реалізації проекту можна виділити високу продуктивність рендерингу сторінок, гнучкість в інтеграції з API та

підтримку динамічних оновлень, а також зручність розробки інтерфейсів, що потребують інтерактивності та частого оновлення даних.

Ця архітектура та технологічний стек забезпечують відповідність сучасним стандартам і дозволяють ефективно вирішувати завдання аналізу та моніторингу цін на ігрові предмети.

Такий вибір архітектурного рішення обґрунтовано кількома ключовими перевагами. По-перше, забезпечується висока швидкість роботи: усі елементи завантажуються одразу, а під час виконання дій на сторінці дані просто оновлюються без перезавантаження. По-друге, досягається значна економія часу, оскільки більшість операцій виконується у браузері, що зменшує кількість звернень до сервера. Нарешті, архітектура забезпечує високу продуктивність завдяки оптимізації коду та мінімізації серверних запитів.

Компонентна архітектура: Next.js дозволяє розділити інтерфейс і сторінки на окремі компоненти, що спрощує організацію та підтримку коду, особливо для великих і складних адміністративних інтерфейсів. Next.js також полегшує реалізацію маршрутизації та інтеграцію з API, а також має набір інструментів для захисту компонентів. Крім того, Next.js має активну спільноту розробників, яка забезпечує доступ до різноманітних інструментів, бібліотек і розширень для підтримки розробки SPA веб-застосунків.

Для створення сучасного дизайну UX/UI використовуються SASS, Talewind.

Для розробки серверної частини використовується Python із фреймворком Django. Мікросервісна архітектура дозволяє реалізовувати функціональність за допомогою незалежних компонентів, що підвищує масштабованість і знижує складність підтримки. Для зберігання даних обрано MongoDB, яка забезпечує високу ефективність роботи з великими обсягами інформації. Використання патерну MVC дозволяє легко проектувати й тестувати функціональність, а модуль JPA спрощує взаємодію з базою даних. Для тестування API застосовувався Swagger2.

3.2 Розробка функціональності та структури системи

Для розробки веб-застосунку системи моніторингу динаміки цін для ігрових предметів було використано популярний патерн проектування, що базується на принципі моделі MVC. MVC дозволяє відокремити логіку програми від її представлення та взаємодії з даними, спрощуючи розробку, тестування та обслуговування програмного забезпечення. Архітектура MVC у Next.js реалізується наступним чином:

Модель: У Next.js дані можуть бути представлені у вигляді об’єктів або класів, які містять необхідні властивості та методи. Це забезпечує структурування та зручність роботи з інформацією, такою як ціни, параметри ігрових предметів або історія змін.

Представлення: Next.js підтримує шаблони JSX для динамічного відображення даних моделі, інтерактивність яких можна налаштувати за допомогою бібліотек, таких як Next UI і Tailwind CSS. Представлення відповідає за взаємодію з користувачем: відображення даних, збір введення або навігацію між сторінками. Дизайн можна легко адаптувати або змінити за допомогою стилізації через SASS або Tailwind CSS.

Контролер: У Next.js контролерами виступають компоненти, які діють як посередники між моделями та представленнями. Контролери отримують дані через API-запити за допомогою Axios, обробляють їх і передають у представлення для відображення. Крім того, компоненти можуть відстежувати зміни даних у реальному часі за допомогою RxJS і відповідно оновлювати представлення.

На основі принципу патерну MVC і компонентної архітектури програми сформовано структуру клієнтської системи. Організація структури папок у проекті була розроблена з урахуванням гнучкості та зручності роботи над кожним із функціональних модулів. У таблиці 3.1 представлена організаційна структура папок у проекті.

Таблиця 3.1 – Організаційна структура папок

Назва папок	Опис
public	Тут зберігаються статичні файли (webp, svg, ttf, woff, json-анімації), які доступні за прямими посиланнями, наприклад http://localhost:3000/назва_файлу. Це зручно для використання у верстці чи стилях без імпорту в код компонентів.
app	Ця папка забезпечує SSR-рендеринг, навігацію між сторінками без додаткових бібліотек, інтеграцію Redux через store-provider і відображення даних про ціноутворення Steam у підпапці view, де компоненти рендерять таблиці, графіки та списки.
composable	Папка містить набір повторюваних компонентів, таких як кнопки, іконки, випадаючі списки та інпут поля, оформлених у вигляді бібліотеки готових елементів. Це забезпечує узгодженість інтерфейсу та спрощує внесення змін, які автоматично застосовуються у всіх частинах програми.
entities	Цей розділ містить описові моделі даних, DTO для передачі даних між фронтендом і бекендом, а також Enums для зручного використання фіксованих значень, як-от перелік валют чи країн. Також файли з деклараціями та методами для розширення типів JavaScript, що спрощує форматування даних і уникає дублювання функцій у проекті.
service	Цей шар відповідає за інтеграцію з бекендом і мікросервісами, реалізуючи логіку роботи з API за допомогою Axios і RxJS, наприклад, для отримання цін на предмети чи профілі користувачів.
store	Містить redux-слайси кожен з яких відповідає за керування своїм станом застосунку.

Продовження таблиці 3.1

Назва папок	Опис
style	Ця папка містить глобальні стилі, налаштування тем, змінні кольорів і розмірів, доповнюючи утилітарні класи TailwindCSS. Тут зберігаються кастомні CSS-класи, SCSS-змінні та стилі для нестандартних елементів інтерфейсу.
utils	Папка містить допоміжні функції, фільтри, хелпери та константи, такі як форматування дат, математичні операції або URL-адреси сервісів. Використання цієї папки дозволяє уникати дублювання коду і робить проект більш структурованим та зрозумілим.

Після розробки організаційної структури папок, наступним кроком є деталізація структури системи за допомогою бекенд-логіки. Це дозволяє чітко визначити архітектуру системи, взаємодію її компонентів та основні процеси, що забезпечує краще розуміння роботи системи та полегшує її подальшу розробку й підтримку.

Бекенд-логіка визначає основні функціональні модулі, такі як обробка запитів, робота з базою даних, управління користувачами та взаємодія з зовнішніми API. На цьому етапі створюються основні контролери, сервіси та моделі, які забезпечують необхідну функціональність. Правильна організація бекенд-частини гарантує стабільність системи, високу продуктивність і можливість масштабування.

Після цього етапу важливо перейти до розробки бази даних, яка стане основою для зберігання всіх об'єктів, відносин та даних системи. Структура бази даних повинна відповідати вимогам бізнес-логіки та забезпечувати швидкий доступ до необхідної інформації. Визначення таблиць, їх зв'язків, індексів та правил обробки даних є ключовим для оптимальної роботи системи.

База даних є невід’ємною частиною будь-якої системи, оскільки вона забезпечує ефективне зберігання, організацію та управління великими обсягами

даних. Вона дозволяє системі швидко отримувати доступ до потрібної інформації, обробляти її та зберігати результати. Це особливо важливо для складних додатків, таких як система моніторингу цін, де потрібно опрацьовувати дані про товари, ціни, користувачів та інші динамічні параметри.

Для забезпечення ефективного зберігання та обробки даних у системі критично важливо правильно спроєктувати базу даних. Вона слугує основою для зберігання інформації про користувачів, товари, ціни та інші ключові дані, які використовуються у роботі системи. Чітка структура бази даних та продумана схема її взаємодії з бекенд-логікою дозволяють оптимізувати доступ до даних, підтримувати їхню цілісність та забезпечувати швидкість обробки запитів, що є важливим для стабільної роботи системи моніторингу цін. На рисунку 3.1 представлена структура бази даних.

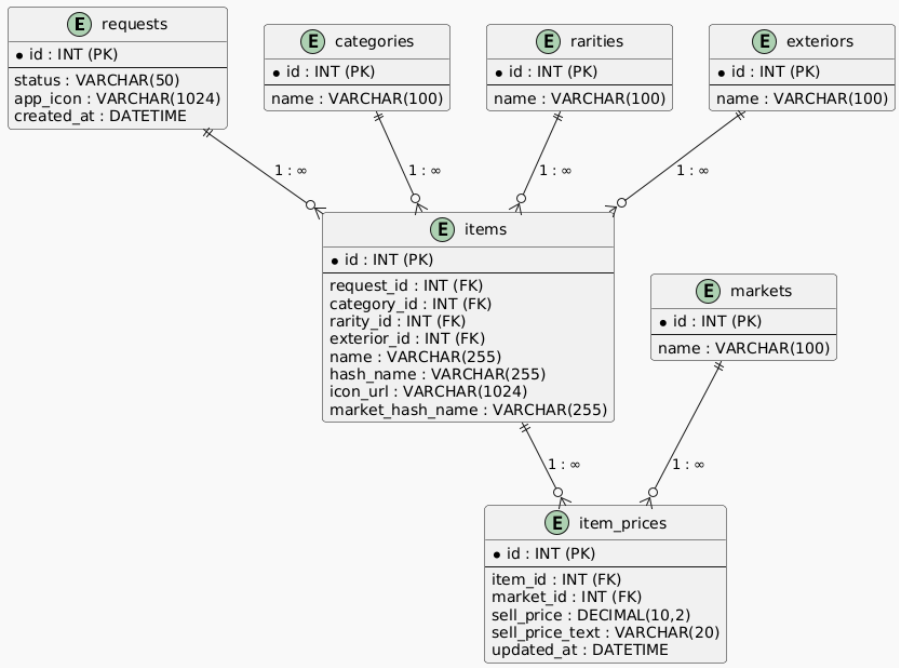


Рисунок 3.1 – Структура бази даних

Після створення структури, важливо перейти до детального аналізу бази даних, оскільки вона відіграє ключову роль у функціонуванні системи. У цьому розділі буде розглянуто основні таблиці, їх взаємозв’язки та типи даних, які зберігаються. Це дозволить зрозуміти, як організовані дані в системі, які процеси

забезпечують їхню узгодженість, та як база даних підтримує основну функціональність додатку. У таблиці 3.2 представлений опис бази даних.

Таблиця 3.2 – Опис таблиць бази даних

Назва таблиці	Опис
requests	Зберігає інформацію про запити до системи. Містить статус, іконку запиту та дату створення.
categories	Містить категорії предметів (наприклад, кейси, наклейки, зброя).
rarities	Зберігає інформацію про рівень рідкості предметів (наприклад, Covert, Classified).
exteriors	Містить інформацію про стан (знос) предметів (наприклад, Factory New, Minimal Wear).
items	Основна таблиця предметів. Зв’язана із запитами, категоріями, рівнями рідкості та станом.
markets	Список торгових платформ, де продаються предмети (наприклад, Steam, DMarket).
item_prices	Зберігає ціни предметів на різних торгових платформах, а також дату останнього оновлення ціни.

Таблиця **requests** містить дані про запити в системі. Вона включає такі поля: id — унікальний ідентифікатор запиту (первинний ключ), status — статус запиту (наприклад, "успішний" або "помилка"), app_icon — URL-адресу іконки, пов’язаної із запитом, та created_at — дату і час створення запиту. Така структура дозволяє зберігати та відстежувати інформацію про всі запити, їхній стан і час виконання

Таблиця **categories** призначена для зберігання даних про категорії предметів. Вона містить два поля: id — унікальний ідентифікатор категорії (первинний ключ) та name — назва категорії, наприклад, "Case" або "Sticker". Така структура забезпечує зручну класифікацію предметів у системі.

Таблиця **rarities** зберігає інформацію про рівні рідкості предметів. Вона включає поля: id — унікальний ідентифікатор рівня рідкості (первинний ключ) та name — назву рівня рідкості, наприклад, "Covert" або "Classified". Це дозволяє системі ефективно організовувати та відображати предмети за їхньою рідкістю.

Таблиця **exteriors** містить інформацію про рівні зносу предметів. Вона включає поля: id — унікальний ідентифікатор рівня зносу (первинний ключ) та name — назву рівня зносу, наприклад, "Factory New" або "Minimal Wear". Це допомагає системі класифікувати предмети за їхнім станом.

Таблиця **items** зберігає інформацію про предмети, що моніторяться системою. Вона містить такі поля: id — унікальний ідентифікатор предмета (первинний ключ), request_id — посилання на запит, з якого отримано дані про предмет (зовнішній ключ до таблиці requests), category_id, rarity_id та exterior_id — зовнішні ключі до таблиць categories, rarities і exteriors, що вказують на категорію, рідкість та рівень зносу предмета відповідно. Також є поля name — назва предмета, hash_name — унікальний хеш для ідентифікації на торгових платформах, icon_url — URL-адреса іконки та market_hash_name — унікальне ім’я на маркетплейсах. Ця структура дозволяє системі зберігати детальні дані про предмети та їх характеристики для подальшого використання.

Таблиця **markets** зберігає інформацію про торгові платформи. Вона містить такі поля: id — унікальний ідентифікатор платформи (первинний ключ) та name — назва платформи, наприклад, "Steam" або "DMarket". Ця структура дозволяє системі підтримувати взаємодію з різними платформами для обміну предметами.

Таблиця **item_prices** зберігає інформацію про ціни предметів на різних платформах. Вона містить такі поля: id — унікальний ідентифікатор запису (первинний ключ), item_id — посилання на предмет, для якого зберігається ціна (зовнішній ключ до таблиці items), market_id — посилання на платформу, де предмет продається (зовнішній ключ до таблиці markets), sell_price — ціна предмета у числовому вигляді, sell_price_text — ціна предмета у текстовому форматі, наприклад, "$1.53", та updated_at — дата та час останнього оновлення

ціни. Ця структура дозволяє відстежувати динаміку цін на різних торгових майданчиках.

Для ефективного моніторингу цін та управління даними в системі необхідно чітко визначити зв’язки між основними таблицями. Залежності між запитами, предметами, категоріями, рівнями рідкості, зносом та цінами дозволяють створити логічну структуру, що забезпечує організоване зберігання інформації та підтримку різноманітних функцій системи.

requests → items: Один запит може містити багато предметів. (1:N)

Це означає, що кожен запит пов’язаний з кількома предметами, які були отримані в процесі моніторингу.

categories → items: Кожен предмет належить до однієї категорії. (1:N)

Тобто кожен предмет має лише одну категорію, яка визначає його тип (наприклад, "Case", "Sticker").

rarities → items: Кожен предмет має одну визначену рідкість. (1:N)

Кожен предмет асоціюється з рівнем рідкості, таким як "Covert" або "Classified".

exteriors → items: Кожен предмет має один рівень зносу. (1:N)

Рівень зносу визначає стан предмета — наприклад, "Factory New" або "Minimal Wear".

items → item_prices: Один предмет може мати багато цін на різних платформах. (1:N)

Кожен предмет може бути представлений з різними цінами на різних торгових майданчиках.

markets → item_prices: Кожна ціна прив’язана до однієї торгової платформи. (1:N)

Кожна записана ціна має пов’язання з конкретною платформою, де ця ціна актуальна.

Ця структура забезпечує правильну організацію даних, уникаючи дублювання, та дозволяє легко масштабувати базу даних у майбутньому.

3.3 Функціонал клієнтської частини системи

Клієнтська частина системи відповідає за взаємодію користувачів із застосунком, забезпечуючи зручний інтерфейс для перегляду, пошуку та обробки інформації про предмети та ціни. Основною метою є створення інтуїтивно зрозумілого користувацького досвіду, який дозволяє ефективно працювати з даними, що надаються системою.

На рисунку 3.2 зображений інтерфейс клієнтської частини системи, який представляє список предметів із відповідною інформацією про ціни та різницю між ними.

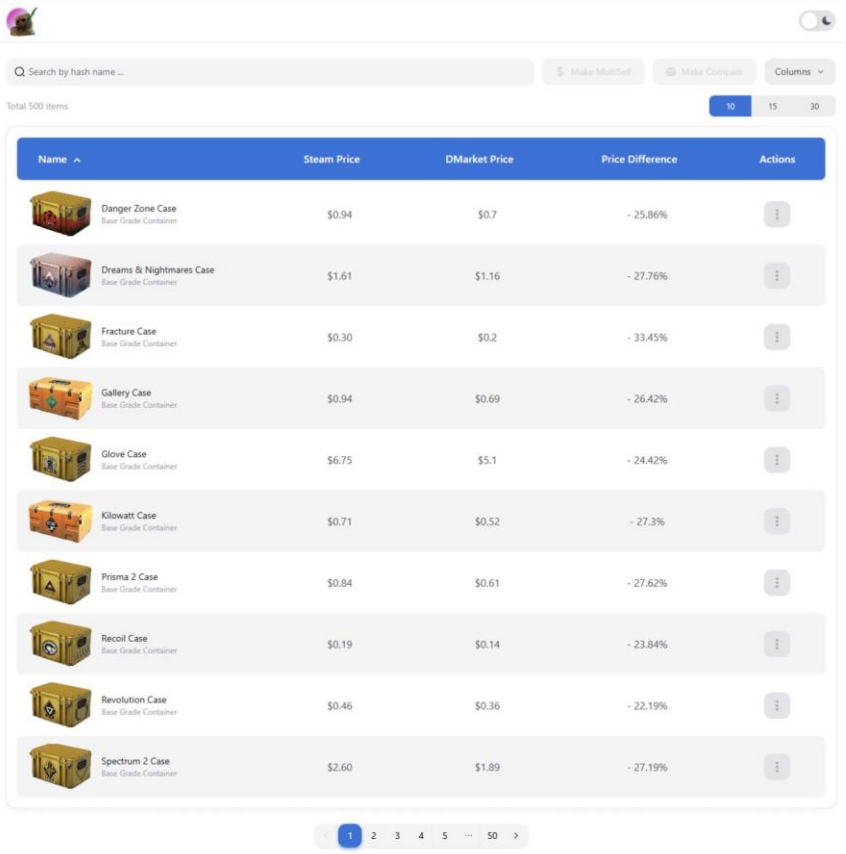


Рисунок 3.2 – Головна сторінка системи

Пошуковий рядок (Рис 3.3), розташований у верхній частині інтерфейсу, дозволяє користувачам швидко знайти потрібний предмет за його назвою чи хеш-ідентифікатором. Цей інструмент оптимізує процес навігації по великому каталогу, що містить багато елементів. Завдяки інтерактивності пошуку

користувачі можуть миттєво отримувати релевантні результати, що значно спрощує взаємодію із системою та економить час. Це особливо важливо для тих, хто шукає конкретні предмети для аналізу цін чи виконання торгових операцій.

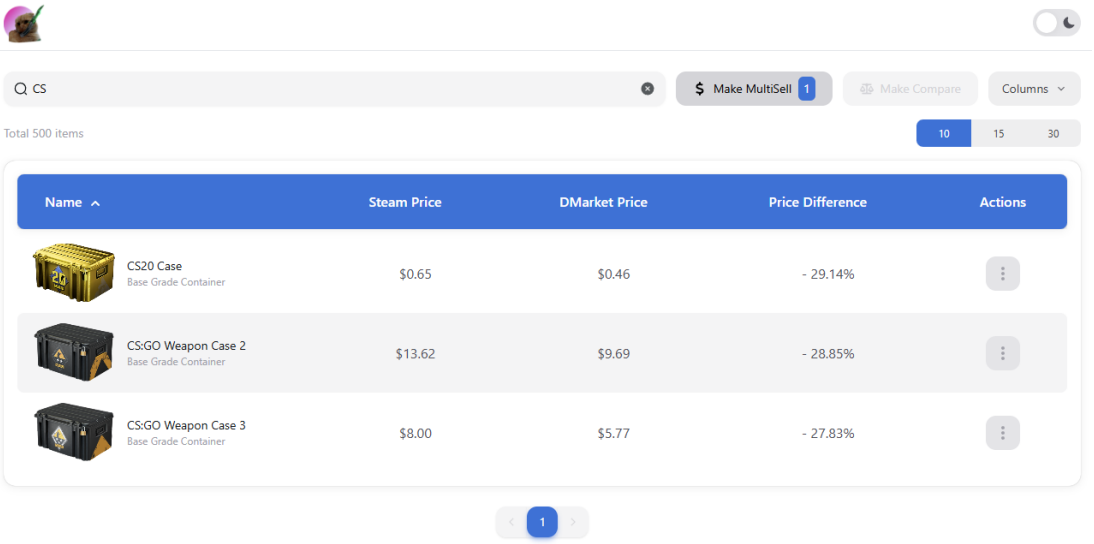


Рисунок 3.3 – Демонстрація процесу пошуку

Нище йде таблиця даних яка забезпечує зручне відображення ключової інформації про предмети та їхні ціни. У колонці Name (Назва) зазначено назву предмета, його мініатюру та категорію (наприклад, "Base Grade Container"), що допомагає користувачам легко ідентифікувати товар. Steam Price (Ціна на Steam) показує актуальну вартість предмета на платформі Steam, яка є базою для подальшого порівняння. У колонці DMarket Price (Ціна на DMarket) вказана ціна того самого предмета на платформі DMarket, що дозволяє оцінити різницю між платформами. Price Difference (Різниця в ціні) відображає відсоткову різницю між цими цінами, дозволяючи швидко визначити, на якій платформі товар вигідніший. Остання колонка, Actions (Дії), містить інтерактивні кнопки, що відкривають можливості для перегляду деталей, додавання до обраного чи здійснення інших операцій із предметами безпосередньо з таблиці.

Таблиця також підтримує функцію сортування(Рис 3.4) , яка дозволяє впорядковувати дані за кожною з колонок. Користувач може сортувати предмети за назвою (Name) для зручного пошуку певного товару, за ціною на платформі

Steam (Steam Price) або DMarket (DMarket Price) для аналізу вартості, а також за відсотковою різницею в ціні (Price Difference) для швидкого виявлення найвигідніших пропозицій. Сортування забезпечує гнучкість у роботі з таблицею, дозволяючи налаштовувати відображення даних під конкретні потреби користувача.

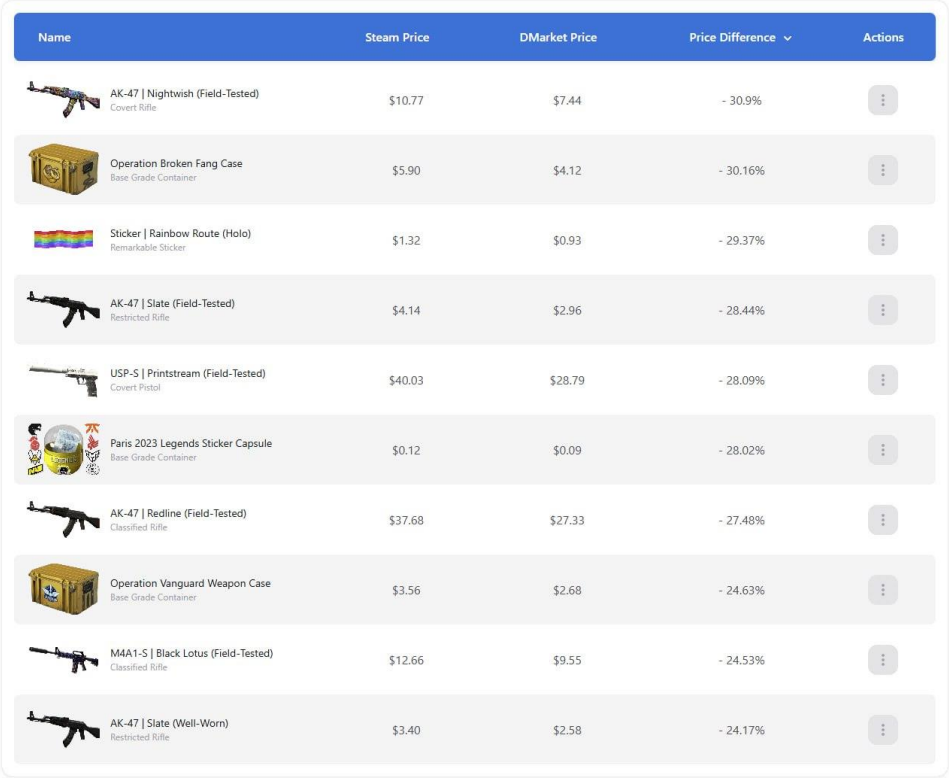


Рисунок 3.4 – Демонстрація процесу сортування

На рисунку 3.5 представлено зображення попапу, який викликається через кнопку Actions для управління вибраним предметом.

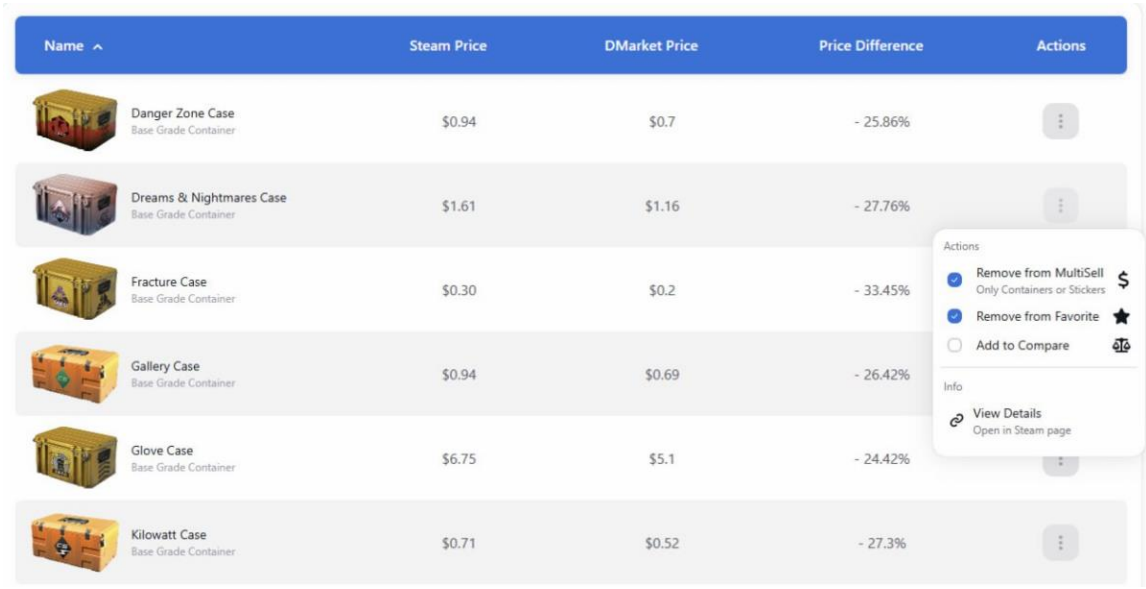


Рисунок 3.5 – Виклик попапу actions

Попап містить два основних розділи: Дії та Інформація. У розділі Дії доступні функції, такі як видалення зі списку масового продажу, виключення зі списку обраних або додавання до порівняння. Розділ Інформація пропонує перегляд детальних даних про предмет на сторінці Steam. Цей інтерфейс забезпечує швидкий доступ до всіх ключових дій для зручного управління предметами.

Кнопка мультисел дозволяє користувачеві запускати процес продажу кількох предметів із їхнього інвентаря. Кнопка динамічно відображає кількість обраних предметів, а також враховує поточний стан списку для визначення, чи доступна дія продажу.

Коли користувач натискає кнопку, вона генерує посилання яке зображено на рисунку 3.6 , що включає інформацію про всі обрані предмети, і відкриває його в новій вкладці браузера, де користувач може обрати кількість і вартість предмету. Після цього список предметів очищується автоматично. Стан списку предметів і доступність кнопки постійно синхронізуються з глобальним сховищем даних, що забезпечує точність і зручність у роботі.

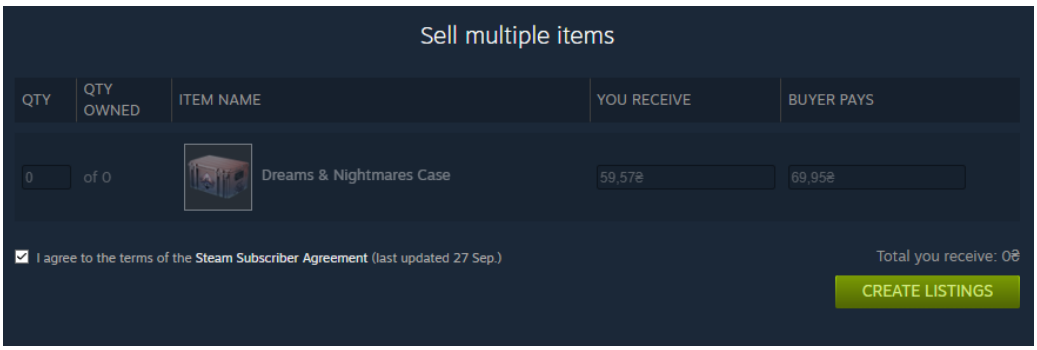


Рисунок 3.6 – Згенерований запит кнопкою мультисел

Кнопка favorite реалізує управління списком обраних предметів у додатку. Список організований за додатками, де для кожного з них можна зберігати до 50 обраних предметів. Користувач може додавати або видаляти предмети зі списку обраного, причому якщо кількість предметів досягає максимального значення, нові не можуть бути додані без попереднього видалення. Код також забезпечує перевірку, чи є предмет у списку, чи активна кнопка додавання, і дозволяє отримувати поточний список обраних для кожного додатка. Усе це інтегровано через глобальне сховище, що дозволяє динамічно керувати станом додатку.

ВИСНОВКИ

Було розглянуто основні поняття веб-додатків, системи моніторингу цін для ігрових предметів на P2P маркетах, а також методи отримання актуальної інформації через веб-скрейпінг та API-інтеграції. Було досліджено важливість цих технологій у контексті створення ефективної системи для моніторингу цін, яка забезпечує швидкий доступ до актуальної інформації.

Для проектування системи було прийнято рішення вибрати мову програмування Python та фреймворк Next.js. Вибір фреймворку Next.js обґрунтовано його потужними можливостями для створення швидких і гнучких веб-додатків, а також ефективною підтримкою компонентної архітектури та адаптацією до динамічних змін стану додатку. Мова програмування Python була вибрана з метою забезпечення високої продуктивності та можливості ефективної обробки запитів на бекенді системи.

Для фронтенду було використано Next.js разом із Next UI та Tailwind CSS. Використання Next.js дозволяє створювати швидкі та масштабовані веб-додатки з підтримкою мікросервісної архітектури, що є ідеальним для моніторингу динаміки цін.

Мікросервісна архітектура, обрана для цієї системи, дозволяє легко інтегрувати різноманітні сервіси, такі як API для веб-скрейпінгу, оброблення даних, аналітичні модулі та управління станом додатку. Завдяки цьому користувачі можуть отримувати актуальну інформацію про динаміку цін на ігрові предмети з різних джерел і в зручному інтерфейсі.

У ролі бекенду було використано Python, який забезпечує гнучкість, ефективне управління ресурсами та швидкість обробки даних. Для фронтенду застосовано Next.js з Tailwind CSS, що дозволяє створювати сучасний та інтуїтивно зрозумілий користувацький інтерфейс. Next UI додає додаткові можливості для компонування інтерфейсу та управління темами.

Таким чином, система для моніторингу динаміки цін на ігрові предмети поєднує сучасні фронтенд та бекенд технології, забезпечуючи зручний і ефективний інструмент для відстеження цін і прийняття обґрунтованих рішень на основі актуальної інформації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Avan.market: веб-сайт. URL: https://avan.market.
2. Buff.163: веб-сайт. URL: https://buff.163.com.
3. CS.MONEY: веб-сайт. URL: https://cs.money.
4. Dmarket: веб-сайт. URL: https://dmarket.com.
5. Frank Zammetti. Practical React Native: Build Two Full Projects and One Full Game Using React Native: підручник, 2018. 76 с.
6. Hay L. Researching UX: Analytics: підручник, SitePoint Pty. Ltd, 2017. 106 с.
7. Hevodata [Електронний ресурс]. MongoDB Storage for Efficient Structuring of Data Simplified 101: веб-сайт. URL: https://hevodata.com/learn/mongodb-storage/.
8. INTERNATIONAL RESEARCH JOURNAL OF MULTIDISCIPLINARY STUDIES / Prof. Joshi J.M., Dr. G.M. Dumbre. Basic Concept of E-Commerce: підручник.
9. Introducing DMarket API for Automated Trading: веб-сайт. URL: https://dmarket.com/blog/dmarket-api-for-automated-trading/.
10. MongoDB Documentation. MongoDB: веб-сайт. URL: https://www.mongodb.com/docs/.
11. MySQL Documentation. MySQL: веб-сайт. URL: https://dev.mysql.com/doc/.
12. Niederst Robbins. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics: підручник, 2018. 10 с.
13. O’Reilly. RESTful web services: Web services for the real world. L. Richardson, S. Ruby: підручник, 2007.

14. Official Steam API Documentation: веб-сайт. URL: https://partner.steamgames.com/doc/webapi.
15. Pleskach V.L., Zatonatska T.G. Електронна комерція: підручник. Київ: Знання, 2007. 535 с.
16. PostgreSQL Documentation. PostgreSQL: веб-сайт. URL: https://www.postgresql.org/docs/.
17. Richardson, C. Microservices patterns: With examples in Java. Manning: підручник, 2018.
18. Skinbaron: веб-сайт. URL: https://skinbaron.de/en.
19. Smartphone Market Size, Share, Growth | Analysis Report. Fortune Business Insights: веб-сайт. URL: https://www.fortunebusinessinsights.com/industry-reports/smartphone-market-100308.
20. SQLite Documentation. SQLite: веб-сайт. URL: https://www.sqlite.org/docs.html.
21. STEAM market: веб-сайт. URL: https://steamcommunity.com/market/.
22. Tutorialspoint. MongoDB - Data Modelling: веб-сайт. URL: https://www.tutorialspoint.com/mongodb/mongodb_data_modeling.htm.
23. Vercel. Next.js: The React Framework for Production: веб-сайт. URL: https://nextjs.org.
24. What Is Web Scraping? How To Legally Extract Web Content: веб-сайт. URL: https://kinsta.com/knowledgebase/what-is-web-scraping/.
25. Wikipedia. База даних: веб-сайт. URL: https://uk.wikipedia.org/wiki/База_даних.
26. Wikipedia. Мікросервіси: веб-сайт. URL: https://uk.wikipedia.org/wiki/Мікросервіси.

27. Understanding Peer-to-Peer, Two-Sided Digital Marketplaces: Pricing Lessons from Airbnb in Barcelona: веб-сайт. URL: https://www.mdpi.com/2071-1050/12/13/5229.
28. Azat Mardan. React Quickly: Painless Web Apps with React, JSX, Redux and GraphQL: підручник, 2017. 18 с.

ДОДАТОК А ФРАГМЕНТИ ПРОГРАМНОГО КОДУ

```
'use client';

import "./table-layout.scss"

import React, {Key, SVGProps, useCallback, useMemo, useState} from "react";

import {

    Button,

    Dropdown,

    DropdownItem,

    DropdownMenu,

    DropdownTrigger,

    Input,

    Selection,

    SortDescriptor,

    Spinner,

    Table,

    TableBody,

    TableCell,

    TableColumn,

    TableHeader,

    TableRow,

    User,

} from "@nextui-org/react";

import response from '@/new_response.json' assert {type: 'json'};

import {ChevronDownIcon, SearchIcon} from "@nextui-org/shared-icons";

import {AppItemEntity} from "@/entities/AppItemEntity";

import TableTopContent from "@/app/view/components/TableTopContent";

import TableBottomContent from "@/app/view/components/TableBottomContent";
```

```
import TableRowActions from "@/app/view/components/TableRowActions";
import BtnAddToFavorite from "@/app/view/components/buttons/BtnAddToFavorite";
import {CompareIcon} from "@/composable/icons/icons";
import {useDispatch} from "react-redux";
import {setSteamAppInfo} from "@/store/slice/steamAppInfoSlice";
import BtnMakeMultiSell from "@/app/view/components/buttons/BtnMakeMultiSell";


export type IconSvgProps = SVGProps<SVGSVGElement> & {
   size?: number;
};


export function capitalize(s: string) {
   return s ? s.charAt(0).toUpperCase() + s.slice(1).toLowerCase() : "";
}


export const columns = [
   {name: "#", uid: "index", applySortable: true, applyFilter: false},
   {name: "Name", uid: "hash_name", sortable: true, applyFilter: false, applyResize: true},
   {name: "Steam Price", uid: "sell_price", sortable: true, applyFilter: true, applyResize: true},
   {name: "DMarket Price", uid: "sell_price_dmarket", sortable: true, applyFilter: true},
   {name: "Price Difference", uid: "sell_price_difference", sortable: true, applyFilter: true},
   {name: "Favorite", uid: "favorite"},
   {name: "Actions", uid: "actions"},
];


const INITIAL_VISIBLE_COLUMNS = ["hash_name", "sell_price", "sell_price_dmarket", "sell_price_difference", "actions"];
```

```
const TableView = () => {
    const dispatch = useDispatch();
    const {data, ...rest} = response

    dispatch(setSteamAppInfo(rest))

    const [filterValue, setFilterValue] = useState("");
    const [visibleColumns, setVisibleColumns] = useState<Selection>(
        new Set(INITIAL_VISIBLE_COLUMNS),
    );
    const [page, setPage] = useState(1);
    const [rowsPerPage, setRowsPerPage] = useState(10);
    const hasFilterApply = Boolean(filterValue);

    const [sortDescriptor, setSortDescriptor] = useState<SortDescriptor>({
        column: "hash_name",
        direction: "ascending",
    });

    const filteredItems = useMemo(() => {
        let filteredUsers = [...data];

        if (hasFilterApply) {
            filteredUsers = filteredUsers.filter((user) =>
                user.name.toLowerCase().includes(filterValue.toLowerCase()),
            );
```

```
    }

    return filteredUsers;
}, [data, filterValue]);

const pages = Math.ceil(filteredItems.length / rowsPerPage);
const headerColumns = useMemo(() => {
    if (visibleColumns === "all") return columns;

    return columns.filter((column) => Array.from(visibleColumns).includes(column.uid));
}, [visibleColumns]);

const items = useMemo(() => {
    const start = (page - 1) * rowsPerPage;
    const end = start + rowsPerPage;

    return filteredItems.slice(start, end);
}, [page, filteredItems, rowsPerPage]);

const sortedItems = useMemo(() => {
    return [...items].sort((a: AppItemEntity, b: AppItemEntity) => {
        const first = a[sortDescriptor.column as keyof AppItemEntity] as number;
        const second = b[sortDescriptor.column as keyof AppItemEntity] as number;
        const cmp = first < second ? -1 : first > second ? 1 : 0;

        return sortDescriptor.direction === "descending" ? -cmp : cmp;
    });
```

```
}, [sortDescriptor, items]);


const renderCell = useCallback((item: AppItemEntity, columnKey: Key) => {
    const cellValue = item[columnKey as keyof AppItemEntity];
    switch (columnKey) {
        case "hash_name":
            return (
                <User
                    avatarProps={
                        {
                            radius: "lg",
                            src: item.icon_url,
                            size: "lg",
                            className: "case-avatar bg-transparent",
                            style: {
                                fontSize: "2rem",
                                aspectRatio: "3/1"
                            }
                        }
                    }
                    description={item.item_type}
                    name={cellValue}
                >
                    {item.item_type}
                </User>
            );
        case "sell_price":
```

```
                return (
                    <p className="text-bold text-medium text-default-600 capitalize">{item.sell_price_text}</p>
                );
            case "sell_price_dmarket":
                return (
                    <p className="text-bold text-medium text-default-600 capitalize">{item.sell_price_dmarket_text}</p>
                );
            case "sell_price_difference":
                return (
                    <p className="text-bold text-medium text-default-600 capitalize">- {cellValue}%</p>
                );
            case "actions":
                return (
                    <TableRowActions item={item}/>
                )
            case "favorite":
                return (
                    <BtnAddToFavorite/>
                )
            default:
                return cellValue;
        }
    }, []);

    const onRowsPerPageChange = useCallback((value: number) => {
        setRowsPerPage(value);
```

```
    setPage(1);
}, []);

const onSearchChange = useCallback((value?: string) => {
    if (value) {
        setFilterValue(value);
        setPage(1);
    } else {
        setFilterValue("");
    }
}, []);

const onClear = useCallback(() => {
    setFilterValue("");
    setPage(1);
}, []);


const classNames = React.useMemo(
    () => ({
        wrapper: ["max-h-[64.5rem] gap-4"],
        th: ["h-16 bg-primary text-base text-primary-foreground shadow-lg px-8"]
    }),
    [],
);

return (
```

```
<div className="flex flex-col text- gap-4 p-4">
  <div className="flex justify-between gap-3 items-end">
    <Input
      isClearable
      size={"md"}
      className="w-full"
      classNames={{
        input: ["px-20"]
      }}
      placeholder="Search by hash name ..."
      startContent={<SearchIcon/>}
      value={filterValue}
      onClear={() => onClear()}
      onValueChange={onSearchChange}
    />
    <div className="flex gap-3">
      <BtnMakeMultiSell/>
      <Button disabled={true} className={"disabled:bg-default-100 disabled:text-default-300 px-5"}
              startContent={<CompareIcon size={16}/>}>
        Make Compare
      </Button>
      <Dropdown>
        <DropdownTrigger>
          <Button endContent={<ChevronDownIcon className="text-small"/>} variant="flat">
            Columns
          </Button>
```

```
            </DropdownTrigger>
            <DropdownMenu
              disallowEmptySelection
              aria-label="Table Columns"
              closeOnSelect={false}
              selectedKeys={visibleColumns}
              selectionMode="multiple"
              onSelectionChange={setVisibleColumns}
            >
              {columns.map((column) => (
                <DropdownItem key={column.uid} className="capitalize">
                  {capitalize(column.name)}
                </DropdownItem>
              ))}
            </DropdownMenu>
          </Dropdown>
        </div>
      </div>
      <Table
        isStriped
        classNames={classNames}
        isHeaderSticky={true}
        aria-label="item table view"
        bottomContent={
          <TableBottomContent itemTotal={data.length} hasFilterApply={hasFilterApply}
                              setPageSize={setPage} currentPage={page} totalPages={pages}
          />
```

```
    }
    topContent={
      <TableTopContent itemTotal={data.length} filterValue={filterValue}
                       hasFilterApply={hasFilterApply}
                       onSearchChange={onSearchChange}
                       rowsPerPage={rowsPerPage}
                       onRowsPerPageChange={onRowsPerPageChange}
                       visibleColumns={visibleColumns}
      />
    }
    bottomContentPlacement="outside"
    topContentPlacement="outside"
    sortDescriptor={sortDescriptor}
    onSortChange={setSortDescriptor}
  >
    <TableHeader columns={headerColumns}>
      {(column) => (
        <TableColumn
          key={column.uid}
          align={column.uid !== "hash_name" ? "center" : "start"}
          allowsSorting={column.sortable} allowsResizing={true}
        >
          {column.name}
        </TableColumn>
      )}
    </TableHeader>
    <TableBody items={sortedItems} loadingContent={<Spinner label="Loading..."/>}>
```

```
                {(item) => (
                  <TableRow key={item.name} className={`tr-hover`}>
                    {(columnKey) => (
                      <TableCell>{renderCell(item, columnKey)}</TableCell>
                    )}
                  </TableRow>
                )}
              </TableBody>
            </Table>
          </div>
    );
}



export default TableView
```

ДОДАТОК Б ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ(презентація)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
на ступінь вищої освіти магістр
із спеціальності 122 Комп'ютерні науки

СИСТЕМА ДЛЯ МОНІТОРИНГУ ДИНАМІКИ ЦІН ДЛЯ ІГРОВИХ ПРЕДМЕТІВ НА P2P МАРКЕТАХ НА ОСНОВІ ВЕБ-СКРЕЙПІНГУ ТА API-ІНТЕГРАЦІЙ

ВИКОНАВ: СТУДЕНТ 6 КУРСУ, ГРУПИ КНДМ-61 ШУШВАЛ РОМАН ВІКТОРОВИЧ

КЕРІВНИК РОБОТИ: ВАСИЛЕНКО В. В. К. Т. Н., ДОЦ.

Київ - 2024

Слайд 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Тема	Система для моніторингу динаміки цін для ігрових предметів на P2P маркетах на основі веб-скрейпінгу та API-інтеграцій
Мета дослідження	створення прототипу системи для моніторингу динаміки цін ігрових предметів на P2P маркетах
Наукове завдання	розробка прототипу системи для моніторингу динаміки цін для ігрових предметів на P2P маркетах на основі веб-скрейпінгу та API-інтеграцій
Об'єкт дослідження	процес аналізу даних та їх візуалізації на основі динаміки цін ігрових предметів
Предмет дослідження	інструменти та технології автоматизації аналізу та візуалізації даних, включаючи методи веб-скрейпінгу та API-інтеграцій.

Слайд 2

АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ТЕХНОЛОГІЙ

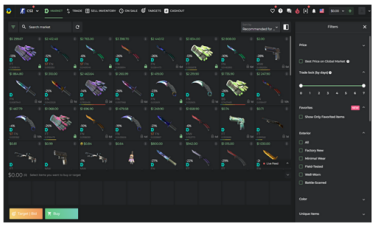
Сучасні підходи до моніторингу цін ігрових предметів мають на меті забезпечити глибоке розуміння ринку, допомагаючи гравцям та трейдерам отримувати актуальну інформацію про вартість різних об'єктів у відеоіграх. Серед інструментів, які широко використовуються, є P2P (Peer-to-Peer) маркетплейси, які дозволяють прямий обмін товарами між гравцями без участі посередників.

Технології веб-скрейпінгу, такі як Next.js і Talewind, забезпечують ефективне автоматичне збирання та обробку даних з веб-сайтів. **Next.js** дозволяє створювати універсальні веб-додатки з підтримкою серверного рендерингу, а **Talewind** допомагає швидко розробляти адаптивні інтерфейси з сучасним дизайном. Разом вони створюють потужне рішення для збору даних та їх візуалізації.

Слайд 3

ОГЛЯД НАЙПОПУЛЯРНІШИХ P2P МАРКЕТІВ

Steam – найбільший ігровий маркет, який дозволяє обмінюватися предметами між гравцями через Steam Marketplace. Особливістю є велика кількість активних користувачів та широкий асортимент ігрових предметів.



DMarket – спеціалізований маркет для торгівлі цифровими предметами з багатьох ігрових платформ, включаючи CS:GO, Dota 2, та інші. Головною перевагою є підтримка блокчейну, що забезпечує прозорість та захист угод.

Слайд 4

ВИБІР ТЕХНОЛОГІЙ

Для розробки бекенду був використаний Python завдяки його гнучкості, простоті в освоєнні та широкому набору інструментів для обробки даних і створення ефективних серверних рішень. Python дозволяє швидко розробляти стабільні та масштабовані рішення завдяки підтримці безлічі фреймворків, бібліотек і інтеграцій з іншими технологіями.

Для фронтенду було використано Next.js та Talewind. Next.js дозволяє створювати універсальні веб-додатки з підтримкою серверного рендерингу та статичного генерації, що забезпечує високу продуктивність і SEO-оптимізацію. Talewind – це фреймворк для швидкої розробки інтерфейсів користувачів з адаптивним дизайном і сучасним стилем, що полегшує створення привабливих і зручних для користувачів інтерфейсів. Разом ці технології забезпечують потужний фронтенд-рішення.

Слайд 5

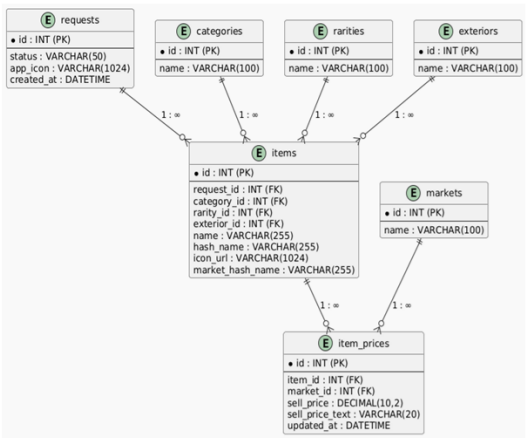
АРХІТЕКТУРА СИСТЕМИ

Клієнтська частина відповідає за інтерфейс користувача, що забезпечує зручну навігацію, перегляд даних та взаємодію з сервісами системи. Вона розроблена з використанням сучасних технологій фронтенду, таких як Next.js для керування маршрутизацією та рендерингом сторінок, а також Tailwind CSS для створення зручного та адаптивного дизайну з акцентом на високій продуктивності та зручному UI/UX..

Серверна частина проекту реалізована з використанням бекенд-логіки для обробки запитів, управління даними та взаємодії з базами даних. Основні компоненти включають контролери, сервіси та моделі, які відповідають за обробку запитів та забезпечення стабільної роботи системи.

Слайд 6

БАЗА ДАНИХ

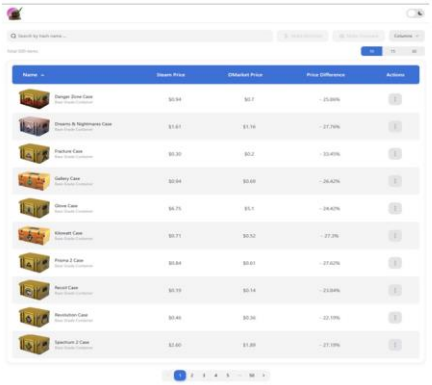


Ця база даних забезпечує структуроване зберігання даних про ігрові предмети, їх властивості та ціни на різних платформах. Використання окремих таблиць для категорій, рівнів рідкості, зносів і цін дозволяє легко масштабувати систему, додавати нові платформи або розширювати властивості предметів.

Слайд 7

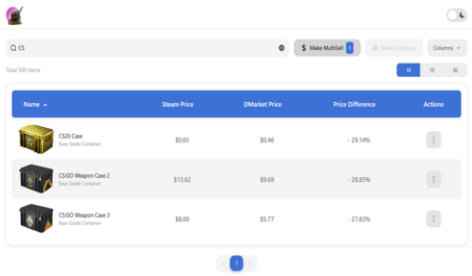
ДЕМОНСТРАЦІЯ СИСТЕМИ

Вигляд головної сторінки системи



Слайд 8

ДЕМОНСТРАЦІЯ СИСТЕМИ

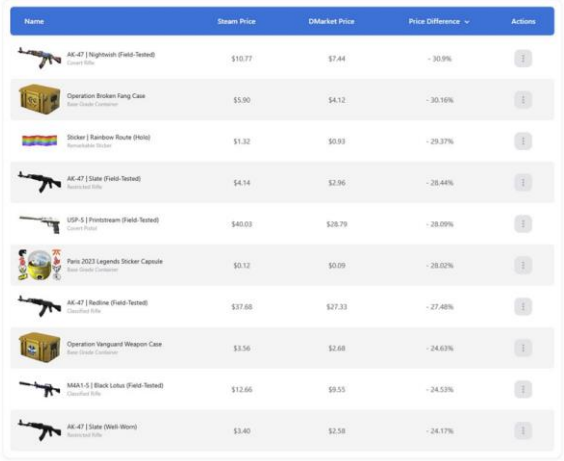


Пошуковий рядок, розташований у верхній частині інтерфейсу, дозволяє користувачам швидко знайти потрібний предмет за його назвою чи хеш-ідентифікатором.

Слайд 9

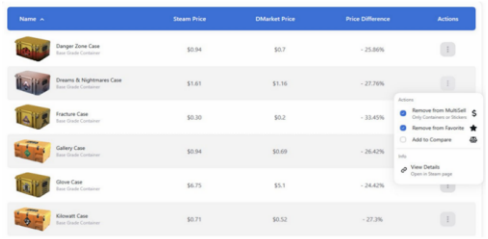
ДЕМОНСТРАЦІЯ СИСТЕМИ

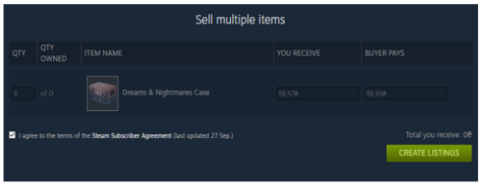
Сортування забезпечує гнучкість у роботі з таблицею, дозволяючи налаштовувати відображення даних під конкретні потреби користувача.



Слайд 10

ДЕМОНСТРАЦІЯ СИСТЕМИ





Кнопка мультисел дозволяє користувачеві запускати процес продажу кількох предметів із їхнього інвентаря. Кнопка динамічно відображає кількість обраних предметів, а також враховує поточний стан списку для визначення, чи доступна дія продажу.
Коли користувач натискає кнопку, вона генерує посилання яке зображено , що включає інформацію про всі обрані предмети, і відкриває його в новій вкладці браузера, де користувач може обрати кількість і вартість предмету

Слайд 11

ВИСНОВКИ

Було розглянуто основні поняття веб-додатків, систем моніторингу динаміки цін для ігрових предметів на P2P маркетах, а також методи отримання актуальної інформації через веб-скрейпінг та API-інтеграції

Для проектування системи було прийнято рішення вибрати мову програмування Python та фреймворк Next.js.

Мікросервісна архітектура, обрана для цієї системи, дозволяє легко інтегрувати різноманітні сервіси, такі як API для веб-скрейпінгу, оброблення даних, аналітичні модулі та управління станом додатку

Було розроблено систему згідно з визначеними вимогами

Слайд 12