МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ ТРЕЙДИНГУ ВІРТУАЛЬНИХ ТОВАРІВ З ПЛАТФОРМИ STEAM

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Віталій БАКУНОВ

«___»_____ 2025 р.

Керівник канд. техн. наук, доцент _____Євген СІДЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп'ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп'ютерні науки
Освітня програма	Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Бакунова Віталія Євгеновича
((прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для трейдингу віртуальних товарів з платформи Steam».

Керівник роботи: Сіденко Євген Вікторович, доцент кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: створення вебзастосунку, який дозволяє здійснювати трейдинг (обмін, купівлю-продаж) віртуальних товарів з платформи Steam. Початковими даними для роботи слугують: вимоги до функціоналу вебзастосунку, особливості API Steam для інтеграції з

платформою, інформація про типи віртуальних товарів (скіни, ключі, предмети тощо), а також вимоги до безпеки транзакцій та захисту даних користувачів.

4. Перелік питань, що підлягають розробці:

– аналіз сучасного стану ринку трейдингу віртуальних товарів із платформи Steam, огляд популярних вебзастосунків для таких операцій;

– вивчення особливостей роботи Steam API для отримання, передачі та обміну інформацією про інвентар та торгові пропозиції;

– розробка архітектури вебзастосунку для трейдингу: вибір технологій, способи інтеграції з Steam, вимоги до безпеки;

– огляд і порівняння різних підходів до забезпечення безпечної аутентифікації та авторизації користувачів (зокрема через Steam);

– розробка та реалізація системи рейтингу й репутації користувачів для забезпечення чесності трейдингових операцій;

– тестування, порівняльний аналіз функціональності, безпеки та зручності користування розробленим вебзастосунком.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віталій БАКУНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для трейдингу віртуальних товарів з платформи Steam.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем, щодо трейдингу віртуальних товарів.	31.01.2025	01.03.2025	
4	Огляд існуючих рішень машинного навчання для вирішення поставленої задачі	02.03.2025	01.04.2025	
5	Реалізація обраних технологій з аналізом отриманих результатів	02.04.2025	15.05.2025	
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	
9	Доробка та остаточне оформлення КР	31.05.2025	10.06.2025	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	11.06.2025	

Керівник роботи _____ Євген СІДЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віталій БАКУНОВ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Бакунова Віталія Євгеновича

на тему: «**ВЕБЗАСТОСУНОК ДЛЯ ТРЕЙДИНГУ ВІРТУАЛЬНИХ ТОВАРІВ З ПЛАТФОРМИ STEAM**»

Актуальність даної роботи визначається стрімким розвитком ринку віртуальних товарів на платформі Steam, зростанням кількості користувачів та необхідністю створення сучасних, безпечних і зручних вебзастосунків для трейдингу цифровими активами. Це обумовлює підвищені вимоги до функціональності, захисту даних та інтеграції із зовнішніми сервісами.

Об'єктом роботи є процес трейдингу віртуальних товарів.

Предметом роботи є програмні інструменти та архітектурні рішення для створення вебзастосунку з трейдингу віртуальних товарів з платформи Steam.

Метою роботи є розробка ефективного, захищеного й функціонального вебзастосунку для трейдингу віртуальними товарами з інтеграцією зі Steam API та використанням сучасного технологічного стеку.

В результаті виконання роботи було проведено аналіз існуючих платформ, моделей монетизації й систем захисту даних, визначено основні технологічні вимоги до вебсервісів цієї сфери. Описано вибір і впровадження стеку технологій (React, Node.js, MongoDB, Docker, Kubernetes), реалізовано архітектуру платформи, основні функціональні модулі, систему рейтингу, чат-бота та комплексний захист користувацьких даних. Окреслено перспективи розвитку платформи: впровадження систем аналітики, прогнозування цін, підтримка мультиплатформеного інтерфейсу.

Робота складається з чотирьох розділів. У першому розділі проаналізовано предметну область, представлено огляд сучасних рішень та аналогічних систем. Другий розділ присвячений вибору й опису технологій, що використані для розробки платформи. Третій розділ містить результати проектування та реалізації вебзастосунку, а також опис основних функціональних модулів. У четвертому розділі розглянуто перспективи розвитку, аналіз отриманих результатів і можливості масштабування. Загальний обсяг роботи — 84 сторінок. Кваліфікаційна робота містить 1 додаток, 17 рисунків, 6 таблиць та 33 джерел посилань.

Ключові слова: Steam, трейдинг, вебзастосунок, віртуальні товари, цифрові активи, React, Node.js, MongoDB, безпека, інтеграція API, чат-бот.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

Vitalii Bakunov

«WEB APPLICATION FOR TRADING VIRTUAL SKINS FROM THE STEAM»

The relevance of this research is determined by the rapid development of the virtual goods market on the Steam platform, the growing number of users, and the need to create modern, secure, and user-friendly web applications for trading digital assets. This leads to increased requirements for functionality, data protection, and integration with external services.

The object of this work is the process of trading virtual goods.

The subject of this work is the software tools and architectural solutions for creating a web application for trading virtual goods from the Steam platform.

The aim of this work is to develop an efficient, secure, and functional web application for trading virtual goods, integrated with the Steam API and utilizing a modern technology stack.

As a result of the work, an analysis of existing platforms, monetization models, and data protection systems was conducted, and the main technological requirements for web services in this field were identified. The selection and implementation of the technology stack (React, Node.js, MongoDB, Docker, Kubernetes) are described, as well as the development of the platform architecture, main functional modules, rating system, chat-bot, and comprehensive user data protection. The prospects for further platform development are outlined, including the implementation of analytics systems, price prediction, and support for a multiplatform interface.

The thesis consists of four chapters. The first chapter analyzes the subject area and provides an overview of modern solutions and similar systems. The second chapter

devoted to the selection and description of the technologies used in the development of the platform. The third chapter presents the results of the system's design and implementation, as well as descriptions of the main functional modules. The fourth chapter discusses development prospects, analyzes the obtained results, and explores scalability opportunities. The total volume of the work is 84 pages. The qualification thesis contains 1 supplement, 17 figures, 6 tables, and 33 sources.

Key words: Steam, trading, web application, virtual goods, digital assets, React, Node.js, MongoDB, security, API integration, chat-bot.

ЗМІСТ

ВСТУП

Віртуальні товари — це ігрові предмети, скіни, аксесуари, які існують винятково у цифровій формі та набули великої популярності в індустрії відеоігор і онлайн-платформ. Їх торгівля вже давно вийшла за межі простої ігрової активності, перетворившись на самостійний сегмент цифрової економіки. Сучасний ринок віртуальних товарів стрімко зростає, про що свідчить статистика провідних аналітичних агентств: за прогнозами, його обсяг досягне понад 112 мільярдів доларів США вже у 2025 році, демонструючи щорічне зростання на рівні майже 20 відсотків. Такі темпи обумовлені як активним розвитком масових онлайн-ігор, так і появою нових економічних моделей у цифрових екосистемах.

Зростаючий попит на унікальний і персоналізований ігровий досвід стимулює появу тисяч нових предметів, скінів, аватарів і доповнень для гравців з усього світу. Багато користувачів розглядають віртуальні товари як інструмент самовираження, підвищення статусу або навіть як інвестиційну можливість. Це формує потужну онлайн-економіку, де цифрові активи продаються, купуються, обмінюються або колекціонуються на спеціалізованих торгових платформах. Для багатьох геймерів ці ринки стали невід'ємною частиною ігрової культури та соціальної взаємодії.

Стрімке зростання ринку віртуальних товарів підсилюється технологічними інноваціями, серед яких варто виділити впровадження елементів доповненої та віртуальної реальності, розвиток метавсесвітів, а також застосування блокчейн-технологій для створення NFT-активів і децентралізованих ринків. Разом із цим зростає і конкуренція між торговими платформами, що змушує їх впроваджувати нові бізнес-моделі, покращувати безпеку, автоматизувати процеси й підвищувати рівень користувацького сервісу.

У цьому контексті комплексний аналіз сучасних вебзастосунків для трейдингу віртуальними товарами є не просто актуальним, а й необхідним. Такий аналіз дозволяє

зрозуміти не лише, які технології та бізнес-моделі стають найефективнішими, але й які виклики постають перед розробниками у сфері продуктивності, масштабованості, монетизації та безпеки. Особливо важливим є питання вибору оптимального технологічного стеку для створення власного продукту, адже від цього напряму залежить і технічна надійність, і комерційний успіх майбутнього вебзастосунку.

Об’єктом роботи є процес трейдингу віртуальних товарів.

Метою роботи є розробка ефективного, захищеного й функціонального вебзастосунку для трейдингу віртуальними товарами з інтеграцією зі Steam API та використанням сучасного технологічного стеку.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ВЕБЗАСТОСУНКІВ ДЛЯ ТРЕЙДИНГУ

1.1 Огляд основних платформ для трейдингу віртуальних товарів

Steam Community Market. Офіційна торгова платформа Valve для ігрових предметів (працює з грошима з Steam Wallet). На ній користувачі можуть купувати та продавати скіни та інші предмети в іграх, спрямовуючи кошти у свій Steam-гаманець (рис. 1.1).

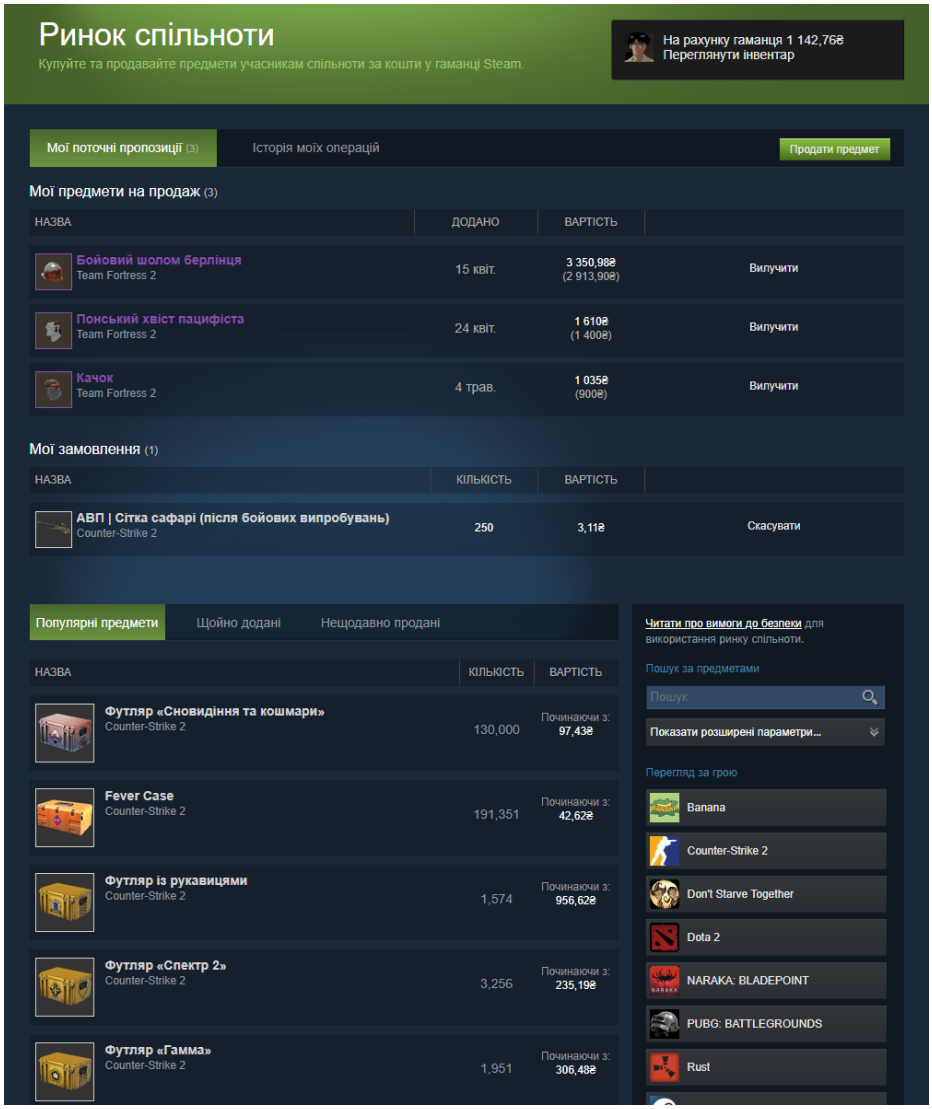

Рисунок 1.1 – Відображення ринку від Valve [1]

Торгівля відбувається централізовано всередині екосистеми Steam; наприклад, на ринку CS:GO чи Dota 2. Комісія Steam Market складається з близько 5% + 10%, тобто у сумі ~15% від ціни продажу (сума не менше $0,01) за кожну транзакцію [1].

CS.MONEY. Одна з найбільш відомих сторонніх платформ для обміну скінами CS2/Dota2. Заснована 2015 року. Підтримує як класичний P2P-формат, так і торгівлю через боти. Модель P2P дозволяє продавцю залишати предмети у власному інвентарі до моменту продажу, але для цього покупець очікує підтвердження угоди. Фактична комісія CS.MONEY становить 7% від суми угоди, що значно нижче за 15%-ву комісію Steam Market, але вища, ніж на деяких інших майданчиках (наприклад, китайському BUFF163 – 2.5%). У 2022 році CS.MONEY запровадила розділ торгового маркету, де користувачі розміщують лоти, а платформа виступає посередником (подібно до ринку Steam) [4].

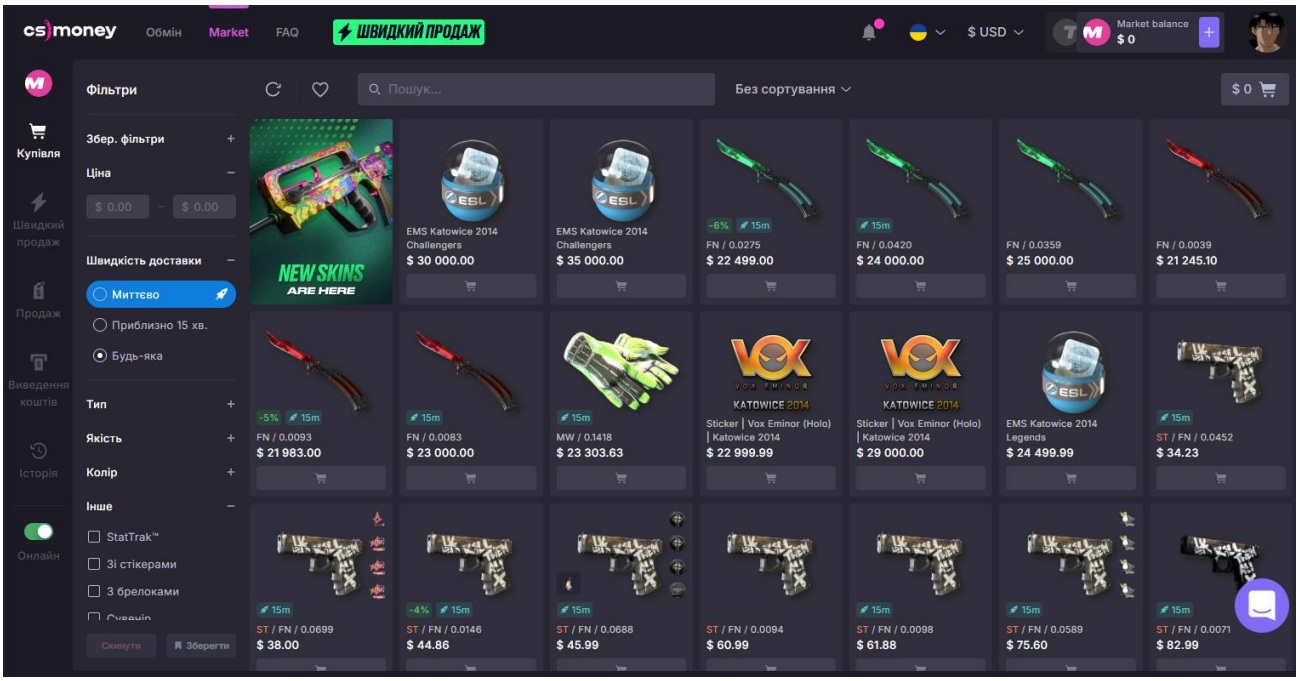

Рисунок 1.2 – Відображення ринку від CS.MONEY [4]

CSFloat. Новий ринок (заснований 2020) спеціально для скінів CS2 (Counter-Strike 2). Працює у форматі P2P (peer-to-peer): користувачі за лот вказують свій предмет і отримують посилання для обміну. Платформа також підтримує оплату через криптовалюти (биткоїн, USDT) для купівлі скінів, що спрощує безпеку платежів. За описом, CSFloat реалізує миттєві обміни з ботами та P2P-торгівлю, намагаючись поєднувати переваги обох підходів [2].

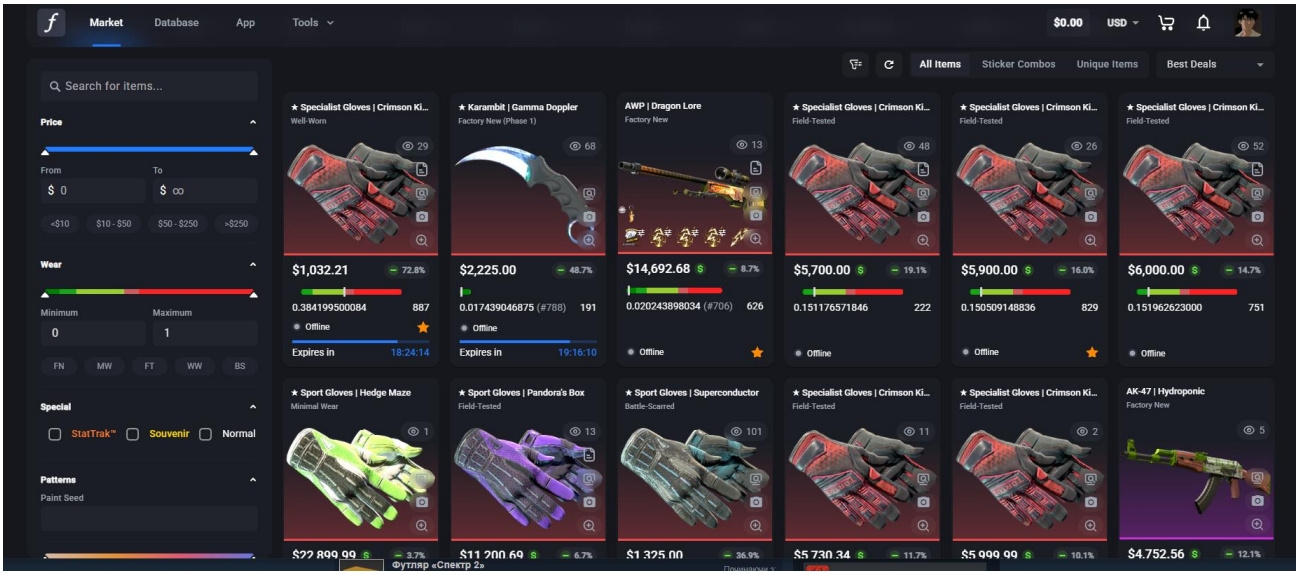

Рисунок 1.3 – Відображення ринку від CSFloat [2]

DMarket. (заснована 2017) Біржа для купівлі/продажу віртуальних предметів з різних ігор. DMarket позиціонує себе як перший глобальний децентралізований маркетплейс віртуальних товарів на базі блокчейну. Тут можна торгувати скінами з CS2, Dota 2, Rust та ін. Модель DMarket – гібрид: з одного боку, технічно це централізована платформа з веб-інтерфейсом; з іншого – внутрішньо вся система базується на смарт-контрактах і блокчейні, що гарантує прозорість обміну. Розробники ігор співпрацюють з DMarket, отримуючи частину комісій за торгівлю своїми предметами. За даними Forbes, на початку діяльності DMarket стягувала комісію 5% від транзакції. На цей час офіційні дані вказують, що комісія за продаж

більшості предметів (CS2, Dota2, TF2, Rust) складає 2–5% залежно від гри. DMarket також приймає криптовалюти до оплати і використовує 2FA (двофакторну аутентифікацію) для безпеки акаунтів [5] [8].

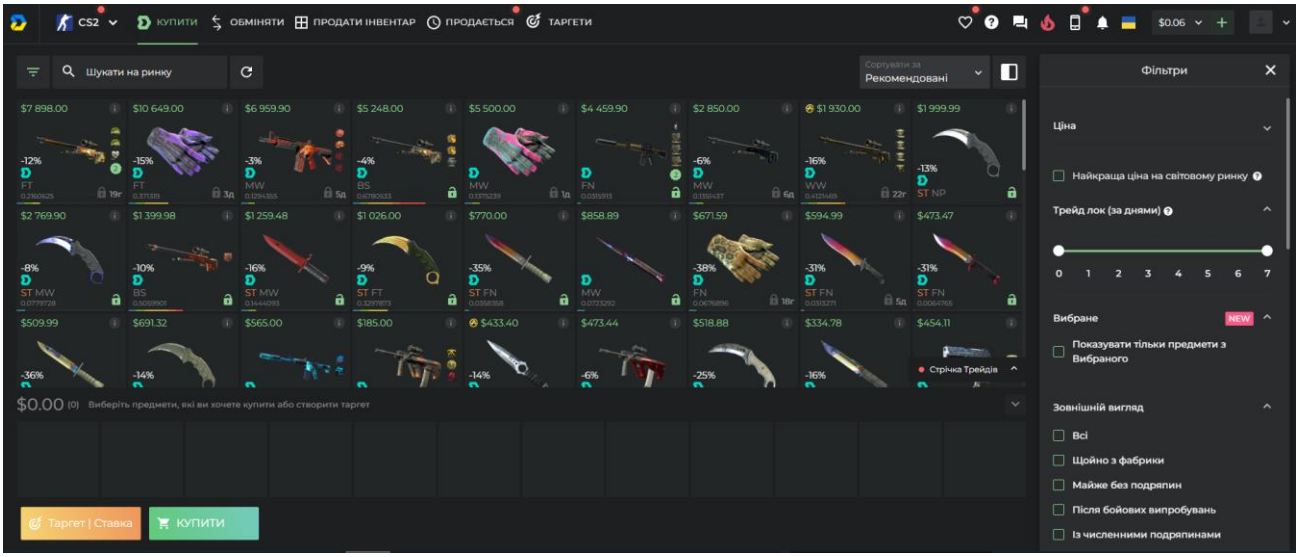

Рисунок 1.4 – Відображення ринку від DMarket [5]

Інші майданчики. Серед українських та міжнародних платформ варто згадати також кейси як Buff163 (китайський ринок скінів з низькою комісією 2.5%), Skinport, Tradeit.gg, LootFarm та ін. Всі вони надають схожий функціонал для популярних ігор (в основному CS2/CS:GO, Dota 2, Rust, TF2). Основні відмінності між платформами лежать у моделі торгівлі (P2P vs боти), крипто-оплаті та внутрішньому UI/UX.

1.2 Функціональні можливості сучасних вебзастосунків

Сучасні вебзастосунки для трейдингу віртуальними товарами стали далеко не просто онлайн-магазинами. Вони перетворилися на багатофункціональні сервіси, що пропонують комплексний досвід і гнучкі інструменти для різних категорій користувачів — від випадкових покупців до професійних трейдерів. Конкуренція на цьому ринку змушує розробників постійно вдосконалювати платформи,

впроваджувати нові технології й забезпечувати максимальну зручність та безпеку операцій.

Ключовим завданням сучасного застосунку є забезпечення швидкого й інтуїтивно зрозумілого пошуку серед тисяч лотів. Для цього платформи впроваджують розумні алгоритми автозаповнення, гнучкі фільтри за назвою, типом, рідкістю, станом предмета чи навіть спеціальними ігровими характеристиками. Продумана система сортування дозволяє легко знаходити найпопулярніші, найновіші або найвигідніші пропозиції, що особливо важливо для активних користувачів, які постійно слідкують за зміною цін.

Не менш важливою є аналітична складова вебзастосунків. Сучасні сервіси дають змогу відстежувати динаміку ринку, будувати графіки змін вартості кожного предмета, переглядати обсяги продажів та аналізувати ринкові тренди. Доступ до історії цін, інтегровані рейтинги, агреговані дані про популярні товари дають змогу користувачам ухвалювати зважені рішення й прогнозувати вигідний момент для купівлі або продажу.

Суттєвою перевагою є можливість налаштовувати сповіщення — наприклад, отримувати повідомлення, коли ціна впаде до бажаного рівня або з’явиться рідкісний лот. Деякі платформи впроваджують рекомендаційні системи на основі машинного навчання: такі сервіси аналізують активність користувача та пропонують індивідуалізовані пропозиції. Завдяки цьому формується не лише зручний інструмент для торгівлі, а й справжня платформа персоналізованої взаємодії.

Важливим елементом є забезпечення прозорості та зручності в управлінні особистим профілем. Користувач може бачити повну історію власних операцій, отримувати фінансові звіти, керувати своїм портфелем та контролювати залишок коштів. Інтеграція із зовнішніми ігровими акаунтами (через Steam OpenID чи OAuth) спрощує вхід, дозволяє автоматично підключати інвентар, а також підвищує рівень безпеки та довіри до платформи [13].

Мобільність є ще одним критично важливим фактором: сучасні платформи розробляють адаптивні інтерфейси, мобільні додатки чи PWA, щоб користувач міг здійснювати всі операції у будь-якому місці та в будь-який час. А додаткові елементи гейміфікації, такі як досягнення, рейтинги, бонуси за активність і внутрішні конкурси, підвищують мотивацію користувачів залишатися на платформі й постійно повертатися.

Можна виділити основні функціональні можливості, що є стандартом для сучасних платформ трейдингу віртуальними товарами:

– інтелектуальний пошук і фільтрація за багатьма параметрами;
– гнучке сортування лотів і підказки у реальному часі;
– інтерактивна аналітика: графіки цін, обсяги продажів, рейтинги;
– налаштовувані сповіщення про зміну вартості, появу лотів, рекомендації;
– детальний лог операцій і управління фінансами через особистий кабінет;
– інтеграція із зовнішніми ігровими платформами для авторизації та автоматичного підключення інвентарю;
– мобільний, адаптивний інтерфейс і підтримка PWA або окремих додатків;
– гейміфікація процесів через бонуси, досягнення, рейтинги і турніри.

Завдяки такому комплексному підходу сучасні вебзастосунки не лише значно спрощують процес трейдингу, а й формують нову культуру онлайн-торгівлі, де користувач отримує максимум сервісу, безпеки й персоналізації незалежно від свого досвіду та обсягу операцій.

1.3 Моделі монетизації: комісії, підписки, реклама

Фінансова модель трейдингових платформ для віртуальних товарів побудована навколо кількох основних джерел доходу, кожне з яких відіграє важливу роль у стабільності та розвитку сервісу. Найбільш традиційним та очевидним способом

заробітку є стягування комісії з кожної успішної угоди. Комісії можуть бути як фіксованими, так і відсотковими, залежно від вартості товару, ліквідності ринку, типу акаунта користувача чи навіть сезону. Так, Steam Community Market застосовує сумарну ставку близько 15%, частину якої отримує розробник гри, а частину — платформа. У свою чергу, сервіси нового покоління, як DMarket або BUFF163, значно знижують відсоток комісій, акцентуючи на масштабності й залученні великої кількості учасників — на цих майданчиках типова ставка становить від 2% до 5%, а іноді й менше для топових трейдерів.

Комісійна модель є зручною для більшості користувачів, оскільки дає змогу легко порівнювати витрати, прогнозувати чистий дохід від продажу та враховувати всі витрати ще до завершення операції. Важливим аспектом стало й те, що у сучасних платформах комісія зазвичай стягується одразу при оформленні угоди, і продавець або покупець відразу бачать фінальну суму до сплати чи отримання. У конкурентних умовах сервіси змушені прозоро інформувати користувачів про структуру комісій та причини зміни ставок, особливо якщо вводяться додаткові збори для екзотичних предметів або малоліквідних позицій.

Поступово монетизація трейдингових платформ перестала обмежуватися лише прямими комісіями. Помітну роль починають відігравати підписки й преміум-членство, які дозволяють активним або професійним трейдерам отримувати ряд переваг: знижені комісійні ставки, розширений функціонал аналітики, доступ до ексклюзивних лотів, прискорене виведення коштів чи навіть пріоритетну підтримку. Наприклад, на CS.MONEY тестується підписка, яка зменшує відсоток комісії для користувача та відкриває нові інструменти для масового розміщення лотів, а на DMarket додаткові функції преміум-доступу дають глибший аналіз ринку та можливість отримувати push-сповіщення про найвигідніші пропозиції в режимі реального часу.

Ще одним напрямом монетизації стали реферальні програми та система бонусів за активність. Власники акаунтів отримують винагороду за запрошення нових користувачів, а також за постійну участь у торгівлі або внесок у розвиток спільноти. Це дозволяє платформі органічно розширювати базу клієнтів, підвищувати залученість та формувати стійке ком'юніті навколо сервісу. Винагороди можуть нараховуватись як у вигляді знижок на комісію, так і у вигляді внутрішньої валюти, яку можна витратити на платформі або обміняти на реальні гроші чи бонуси.

Дещо менш поширеним, але все ж актуальним джерелом доходу залишається реклама. Через специфіку ринку, пряма банерна реклама тут використовується рідко, однак платформи активно співпрацюють з розробниками ігор, кіберспортивними командами та брендами геймерської периферії. Це проявляється у спонсорських акціях, розіграшах, турнірах, спеціальних івентах, інтеграціях партнерських продуктів та маркетингових колаборацій. Деякі сервіси впроваджують нативну рекламу у вигляді «рекомендованих» лотів, партнерських розділів чи ексклюзивних предметів, які просуваються серед аудиторії.

На додачу до традиційних моделей усе частіше впроваджуються платні API та автоматизовані сервіси для професійних трейдерів, які працюють з великими об'ємами даних і вимагають глибокої ринкової аналітики. Платформи поступово відкривають доступ до своїх інтерфейсів за додаткову плату, а також надають користувачам можливість створювати власні дашборди, вести портфелі та підключати сторонні аналітичні сервіси.

Завдяки такому багаторівневому підходу до монетизації платформи мають змогу гнучко балансувати між прибутковістю, лояльністю клієнтів та конкурентною боротьбою за користувача. Це забезпечує стабільний розвиток сервісу навіть в умовах постійного тиску з боку нових гравців і зростаючих витрат на технічне забезпечення та безпеку.

Сучасна модель монетизації трейдингового вебзастосунку — це поєднання гнучкої системи комісій, підписок, бонусів, реферальних та партнерських програм, платних професійних сервісів і співпраці з ігровими брендами, що дозволяє масштабувати бізнес і водночас зберігати лояльність різних категорій користувачів.

1.4 Вимоги до продуктивності та масштабованості

Сучасний вебзастосунок для трейдингу віртуальними товарами має відповідати жорстким вимогам щодо швидкодії, стійкості до навантажень і гнучкості розширення. В умовах, коли тисячі користувачів здійснюють операції водночас, навіть невеликі затримки чи простої можуть призвести до фінансових втрат, негативного досвіду і втрати довіри до сервісу. Особливо це актуально для платформ, що працюють із динамічними ринками, де зміни цін і попиту відбуваються буквально щосекунди.

Висока продуктивність досягається за рахунок продуманої архітектури, використання сучасних технологій розгортання та оптимізації всіх рівнів системи — від бекенду і баз даних до фронтенду і мережевої інфраструктури. Найчастіше критичним стає не лише абсолютний час відповіді сервера, а й здатність системи масштабуватися під час пікових навантажень, залишаючись доступною для всіх користувачів. Для досягнення цього використовують горизонтальне масштабування, коли однакові копії застосунку розгортаються на декількох серверах чи у контейнерах, а балансувальники навантаження рівномірно розподіляють запити між ними. Такий підхід дозволяє збільшувати чи зменшувати кількість ресурсів у реальному часі, підлаштовуючись під зміну аудиторії.

Велика роль належить оптимізації роботи з даними. Кешування (наприклад, через Redis) дає змогу зберігати результати найпопулярніших запитів і зменшувати навантаження на основну базу даних. Реплікація і шардування (розбиття даних на частини) прискорюють виконання складних операцій і дозволяють розподіляти навантаження між багатьма вузлами. Для ресурсоємних завдань — таких як аналіз

великих масивів статистики, побудова графіків чи складні розрахунки — часто використовують фонову обробку в окремих чергах, щоб не блокувати основний потік обробки запитів користувачів.

Значна увага приділяється оптимізації фронтенду. Адаптивний інтерфейс, відкладене завантаження важких компонентів (lazy loading), використання CDN для розповсюдження статичних ресурсів, мінімізація обсягу переданих даних через API (зокрема, впровадження пагінації або GraphQL) — усе це дозволяє забезпечити максимально швидкий відгук незалежно від пристрою користувача чи місця його перебування. Для live-оновлень даних (наприклад, зміни цін у реальному часі чи статусу лота) використовуються WebSocket або подібні технології, що гарантують негайну доставку інформації без перевантаження сервера зайвими запитами.

Стабільність і безперебійна робота під час збоїв досягається шляхом резервного копіювання, автоматичного відновлення після падіння вузлів, а також регулярного моніторингу всієї інфраструктури за допомогою сучасних інструментів (Prometheus, Grafana, Sentry). Це дозволяє команді розробки не лише оперативно виявляти і виправляти проблеми, а й прогнозувати потенційні точки навантаження ще до їх виникнення.

Усі ці підходи реалізуються через сучасні технологічні стекі та сервіси, основні з яких можна виділити так:

– горизонтальне масштабування і контейнеризація: використання Docker і Kubernetes для гнучкого розгортання й автоматичного масштабування додатків, балансування навантаження та відмовостійкості без ручного втручання;

– кешування та оптимізація бази даних: застосування Redis/Memcached для пришвидшення доступу до гарячих даних, шардування та реплікації для розподілу навантаження й підвищення швидкодії при зростанні кількості користувачів;

– асинхронна обробка та черги: винесення складних і важких задач у фонові воркери або окремі черги, щоб основний застосунок залишався швидким і не блокувався;

– оптимізація фронтенду: впровадження CDN, lazy loading, сучасних фреймворків і практик побудови SPA, щоб мінімізувати затримки у взаємодії та швидко доставляти контент на різні пристрої;

– механізми високої доступності: автоматичне резервування й бекапи, регулярний моніторинг усіх компонентів, швидке відновлення після збоїв, підтримка “zero downtime” під час оновлень.

Таблиця 1.1 – Порівняння сайтів

Платформа	Тип торгівлі	Рік запуску	Комісія	Методи оплати	Особливості
Steam Community Market	Централізований (Valve)	2013	15%	Лише Steam Wallet	Офіційний ринок; інтеграція через Steam OpenID; неможливо вивести кошти в реальну валюту
CS.MONEY	P2P + боти	2015	7%	PayPal, Visa, Mastercard, Neteller, Bitcoin, Ethereum, Litecoin	Миттєві обміни ботами; P2P-торгівля; без комісії за виведення; KYC для кешауту на банківську карту
CSFloat	P2P + боти	2020	2%	Банківський переказ, Bitcoin, USDT	Дуже низькі комісії; миттєве виконання угод; підтримка криптовалют

Кінець таблиці 1.1

DMarket	Гібридний (блокчейн + централізований)	2017	2%	PayPal, Visa, Mastercard, Bank Transfer, Cryptocurrency	Смарт-контракти на блокчейні; 2FA; NFT-підтримка; Face2Face P2P-угоди; аналітика через Target-систему

1.5 Безпека та захист користувацьких даних

Безпека — найважливіший аспект для довіри користувачів і стабільної роботи трейдингової платформи. Через фінансову привабливість цифрових предметів цей ринок є мішенню для шахраїв, фішингових атак, ботів і масових зламів. Тому передові платформи впроваджують багаторівневу систему захисту, що охоплює ідентифікацію, шифрування, моніторинг і автоматизоване реагування на інциденти.

Базовим стандартом ідентифікації виступає інтеграція з ігровими акаунтами через Steam OpenID, OAuth або подібні технології, що дозволяє уникнути зберігання паролів та скоротити ризики компрометації. Двофакторна аутентифікація (2FA) є обов’язковою для доступу до акаунту та підтвердження транзакцій, а геолокаційний контроль і перевірка пристроїв допомагають виявляти підозрілу активність [13].

Вся передача даних здійснюється лише через захищені канали (TLS/SSL), а паролі та чутлива інформація зберігаються у зашифрованому вигляді, часто з використанням сучасних алгоритмів типу bcrypt або Argon2. У випадку платіжних та API-операцій використовуються короткоживучі токени з обмеженням прав доступу. Для захисту від фішингу і скаму провідні сервіси, як DMarket, впроваджують спеціальні аналітичні розширення (TrustShield), що автоматично блокують підозрілі сайти і ботів.

Особливу роль у підвищенні прозорості й захисту відіграє використання блокчейн-технологій та смарт-контрактів, які автоматично виконують умови угоди та унеможливлюють зміну записів про транзакції. Система резервування активів — коли і предмет, і гроші блокуються до підтвердження операції обома сторонами — гарантує чесність та мінімізує ризики шахрайства. Автоматичні алгоритми відкату повертають кошти і предмети у разі скасування чи помилки [9].

Щоб додатково підвищити рівень захисту, більшість платформ впроваджують процедури верифікації особи (KYC) для виводу коштів, а також забезпечують постійне інформування користувачів про сучасні методи шахрайства. Технічна й клієнтська підтримка, що працює цілодобово, оперативно реагує на запити щодо блокування акаунтів, підозрілих угод або відновлення доступу. Комплексний багаторівневий підхід до безпеки став не просто стандартом, а умовою виживання сучасної трейдингової платформи у конкурентному середовищі.

Також у торгівлі віртуальними товарами безпека є критично важливою через ризики шахрайства та фішингу. Платформи впроваджують багатошарові заходи захисту:

– *аутентифікація.* Для входу часто використовується Steam OpenID 2.0, що гарантує перевірку справжності аккаунта. Двофакторна аутентифікація (2FA) – обов’язковий атрибут: наприклад, DMarket вимагає при вході пароль та код з мобільного додатку. Це захищає аккаунти навіть у випадку компрометації пароля [13];

– *шифрування.* Усі з’єднання реалізуються через HTTPS/TLS для захисту даних при передачі. Паролі та чутлива інформація зберігаються в зашифрованому вигляді. При можливості використовують OAuth-токени та не зберігають секретних ключів [9];

– *антифішинг і верифікація.* Платформи активного трейдингу мають проблеми з фейковими сайтами і ботами-скамерами. DMarket, наприклад, запустив розширення TrustShield, яке блокує відомі фішингові сайти і неавторизованих ботів,

попереджаючи користувача про підозрілі активності. Таким чином відбувається реальний моніторинг чесності угод [9];

– *блокчейн-технології.* DMarket реалізований на базі блокчейну та смарт-контрактів, що забезпечує нерозривність записів угод. Смарт-контракти автоматизують виконання умов угоди (перевірка наявності предмета та коштів, передача прав власності), що підвищує безпеку системи порівняно з простою централізованою базою даних [9];

– *транзакційний контроль.* При укладанні угоди платформа блокує передачу предмета до підтвердження оплати і навпаки. Уразі розриву угоди (якщо користувач не підтвердив обмін) предмет автоматично повертається власнику, кошти – покупцю. Це знижує ризик шахрайства (наприклад, «закриття торгівлі після отримання предмета»).

Висновки до розділу 1

У першому розділі проведено детальний аналіз сучасного ринку вебзастосунків для трейдингу віртуальних товарів із платформи Steam. Було охарактеризовано основні платформи, їхню функціональність, моделі монетизації, вимоги до продуктивності й масштабованості, а також аспекти безпеки та захисту даних користувачів. Аналіз дозволив визначити ключові тенденції розвитку цієї галузі, а також основні виклики, які постають перед розробниками таких платформ, зокрема забезпечення високої швидкодії, стійкості до навантажень і багаторівневої безпеки. Також було показано, що успішна платформа повинна інтегрувати сучасні інструменти персоналізації, аналітики, гейміфікації та прозорості для забезпечення лояльності та довіри користувачів.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ ТРЕЙДИНГУ

2.1 Фреймворки фронтенду

Розробка сучасного програмного забезпечення для трейдингу віртуальними товарами вимагає комплексного підходу до аналізу вимог, оскільки від точності та повноти їх формулювання залежить не лише якість майбутнього продукту, а й його комерційна привабливість, конкурентоспроможність і технічна надійність. В умовах стрімкого розвитку цифрових ринків і зростання очікувань користувачів визначення вимог стає багаторівневим завданням, що охоплює як функціональні, так і нефункціональні аспекти.

Перш за все, слід враховувати основні сценарії використання майбутньої платформи: реєстрація та авторизація користувачів, зручний пошук і фільтрація лотів, проведення торгових операцій, управління особистим кабінетом, налаштування сповіщень та інтеграція з платіжними і зовнішніми ігровими сервісами. Всі ці дії мають відбуватися швидко, інтуїтивно та безпечно, незалежно від технічної обізнаності користувача. Не менш важливою є потреба у забезпеченні масштабованості, аби платформа могла витримувати зростання навантаження під час пікових подій, таких як ігрові оновлення або великі турніри.

Детальний аналіз вимог охоплює питання безпеки персональних і фінансових даних, необхідність впровадження багатофакторної аутентифікації, використання шифрування під час передавання і зберігання інформації, а також регулярного моніторингу й виявлення потенційних загроз. Досвід провідних платформ свідчить, що саме слабкі місця в архітектурі безпеки стають основною мішенню для зловмисників, тому ці аспекти повинні враховуватися ще на етапі формування вимог.

Особливу увагу варто приділити питанням інтеграції зі сторонніми сервісами. Для сучасної платформи надзвичайно важливо підтримувати авторизацію через

Steam, гаманці, мобільні додатки, а також забезпечувати відкриті API для професійних трейдерів чи партнерів. Ще одним критично важливим напрямком є оптимізація користувацького досвіду — від адаптивного дизайну до швидкості відгуку інтерфейсу й гейміфікації основних функцій, що підвищують лояльність аудиторії.

Для побудови інтерактивного SPA-вебінтерфейсу вибрано React. React – це бібліотека для побудови інтерфейсу користувача з декларативним підходом. Вона використовує концепцію віртуального DOM, яка дозволяє описати бажаний стан UI, а React самостійно оновлює реальний DOM. Це означає, що розробник фокусується на «що показати», а не на механіці роботи з DOM. React підтримує компонентну архітектуру: інтерфейс складається з незалежних модулів (компонентів) з власним життєвим циклом і станом. Це полегшує розробку великих додатків – компоненти можна багаторазово використовувати та ізолювати логіку.

React також пропонує «хуки» (Hooks) – спеціальні функції для роботи зі станом і життєвим циклом в функціональних компонентах. Наприклад, useState для зберігання локального стану, useEffect для виконання побічних ефектів тощо. Такий підхід значно спрощує логіку роботи з даними та запитами до API.

React має багату екосистему бібліотек – наприклад, Redux для глобального стану, React Router для маршрутизації, UI-фреймворки (Material-UI, Ant Design), зручні утиліти (axios, Formik, etc.). Ці інструменти дозволяють швидко реалізовувати необхідний функціонал (списки товарів, форми, діалогові вікна).

Рисунок 2.1 – Логотипи порівнюваних фреймворків [11]

Таблиця 2.1 – Порівняння React з іншими технологіми

Показник	React	Vue.js	Angular
Архітектурний стиль	UI-бібліотека; гнучкість у виборі інструментів	Легка фреймворк-бібліотека; простіше інтегрується	Повноцінний фреймворк з власним DI, формами, HTTP-модулем
Складність навчання	Помірна: необхідно освоїти JSX, хуки, ecosystem	Низька: зрозумілий шаблон, документація перекладена багатьма мовами	Висока: TypeScript, декоратори, складніша структура проекту
Продуктивність	Дуже висока завдяки віртуальному DOM	Трохи поступається, але оптимізує DOM-перехоплення	Може бути нижче через двонаправлений біндинг і важчий runtime
Компонентна архітектура	Підтримує, компоненти чисті, функціональні хуки	Схожа на React, але з меншою екосистемою бібліотек	Підтримує, але компоненти часто великі та важкі через вбудовані функції
Екосистема і спільнота	Дуже широка: безліч пакетів, плагінів, велика кількість вакансій	Широка, але менша за React; зростає швидко	Підтримка Google, але спільнота повільніше росте
Підтримка мобільних	React Native	NativeScript, но менш популярний	Ionic/NativeScript, складніший в налаштуванні
SSR/SSG	Next.js, Gatsby	Nuxt.js	Angular Universal

2.2 Серверні технології

Вибір серверних технологій відіграє вирішальну роль у створенні надійного, масштабованого й продуктивного вебзастосунку для трейдингу віртуальними товарами. Основна мета серверної частини — забезпечити безперервну роботу системи, стабільний обмін даними між клієнтом і базою, а також гарантувати високий рівень безпеки й обробки великої кількості одночасних запитів. У сучасних умовах ринку критично важливо не тільки досягти високої швидкодії, а й створити архітектуру, здатну до гнучкого розширення та стійкості до збоїв.

Однією з найпоширеніших технологій для розробки серверної частини сучасних вебзастосунків є Node.js. Її популярність пояснюється низкою переваг: асинхронною обробкою запитів, широким вибором бібліотек, гнучкістю під час масштабування, а також активною спільнотою. Асинхронна модель Node.js дозволяє ефективно розподіляти навантаження між численними підключеннями, що особливо актуально для платформ з великою кількістю користувачів. Node.js дає можливість швидко розробляти REST API, WebSocket-підключення для реального часу, а також гнучко взаємодіяти із зовнішніми сервісами — від платіжних шлюзів до ігрових API.

Ще одним важливим елементом серверної інфраструктури є використання фреймворків і бібліотек, які пришвидшують розробку та полегшують підтримку коду. Наприклад, такі фреймворки, як Express.js для Node.js, дають можливість організувати чітку структуру серверної логіки, забезпечують маршрутизацію, обробку помилок, логування, взаємодію з базами даних і зовнішніми API. Для побудови гнучких та безпечних систем аутентифікації часто використовують бібліотеки типу Passport.js, які підтримують OAuth, JWT, інтеграцію зі Steam, Google, Facebook тощо. Це дозволяє не тільки підвищити рівень безпеки, а й полегшує процедуру авторизації для користувачів.

Щоб підтримувати масштабованість і гнучке управління ресурсами, дедалі частіше застосовуються технології контейнеризації — Docker і оркестрації Kubernetes. Вони дозволяють розгортати серверні компоненти у вигляді окремих контейнерів, легко масштабувати систему в залежності від навантаження, швидко розгортати оновлення і здійснювати моніторинг роботи всіх сервісів у кластері. Такий підхід підвищує відмовостійкість платформи: у разі збою одного контейнера система автоматично розгортає новий, а балансувальник рівномірно розподіляє запити між усіма доступними копіями застосунку.

Особливе місце у розробці трейдингових платформ займає питання безпеки. Серверна частина повинна забезпечувати захист персональних даних, надійно обробляти фінансові транзакції, а також запобігати шахрайству і спробам зламу. Для цього використовують протоколи HTTPS/TLS, реалізують захист від атак типу SQL-ін’єкцій, CSRF, XSS, впроваджують механізми двофакторної аутентифікації і регулярного моніторингу журналів активності. Окрема увага приділяється розмежуванню прав доступу користувачів, веденню журналів змін і автоматизації резервного копіювання даних.

Підсумовуючи, ключові технології та рішення для сучасної серверної частини трейдингової платформи можна сформулювати так:

– Node.js як основа для швидкої та асинхронної обробки запитів;

– Express.js (або подібний фреймворк) для організації структури додатка й REST API;

– Passport.js, JWT, OAuth для безпечної аутентифікації та авторизації;

– Docker і Kubernetes для контейнеризації, масштабування й стійкості до збоїв;

– обов’язкове використання HTTPS, протоколів шифрування і сучасних стандартів захисту даних;

– регулярний моніторинг, логування, автоматичні бекапи і контроль доступу для забезпечення надійності й захисту.

Рисунок 2.2 – Логотипи порівнюваних фреймворків [17]

Таблиця 2.2 – Порівняння серверних технологій

Показник	Node.js	Python/Django	PHP/Laravel
Мова	JavaScript	Python	PHP
Модель I/O	Неблокуюча, event-driven	Блокуюча, WSGI	Блокуюча, multi-process (PHP-FPM)
Продуктивність	Висока для численних одночасних з’єднань	Середня–висока у CPU-bound задачах	Середня, залежить від PHP-FPM налаштувань
Масштабованість	Горизонтальна з PM2/K8s, кластеризація	Горизонтальна з Gunicorn/uwsgi	Горизонтальна з PHP-FPM/NGINX
Екосистема	npm (1.5M+ пакетів)	PyPI (~350k пакетів)	Composer (~300k пакетів)

Кінець таблиці 2.2

Асинхронна підтримка	Вбудована (async/await, проміси)	Через asyncio, Celery	Обмежена, через сторонні бібліотеки
Фреймворк	Express.js	Django	Laravel
ORM	Mongoose, Sequelize, TypeORM	Django ORM	Eloquent
Інструменти	nodemon, PM2, Webpack	manage.py, Celery, pytest	Artisan, Homestead, Valet
Спільнота	Дуже велика, швидкий розвиток	Велика, стабільний розвиток	Велика, але повільніший розвиток
Типові випадки	Real-time API, чат, стрімінг, мікросервіси	CRUD, аналітика, наукові проекти	CMS, блоги, CRUD-додатки

2.3 Бази даних (MongoDB vs SQL)

При розробці трейдингової платформи для віртуальних товарів вибір бази даних має визначальне значення для продуктивності, масштабованості, гнучкості роботи з даними й подальшого розвитку системи. Сучасний ринок пропонує два основних підходи: використання документно-орієнтованих NoSQL-рішень (наприклад, MongoDB) і класичних реляційних баз даних (таких як PostgreSQL чи MySQL).

MongoDB є одним із найпопулярніших представників NoSQL-технологій. Вона зберігає дані у вигляді документів формату BSON, що дозволяє динамічно змінювати структуру записів і гнучко працювати з неструктурованими або часто змінюваними

даними. Це особливо зручно для зберігання інформації про інвентарі користувачів, угоди, історію змін, де структура запису може відрізнятися від користувача до користувача або розширюватися під час розвитку продукту. MongoDB дозволяє ефективно масштабувати систему за рахунок горизонтального шардування та легко інтегрується з хмарними сервісами, що важливо для проєктів із швидко зростаючим навантаженням.

Класичні SQL-бази даних, зокрема PostgreSQL та MySQL, залишаються стандартом у багатьох фінансових, банківських та трейдингових рішеннях. Їхня головна перевага — чітка структура, підтримка складних транзакцій, гарантована цілісність даних і можливість створення взаємозв'язаних таблиць із суворими обмеженнями типу, унікальності, зовнішніх ключів тощо. Саме завдяки механізмам ACID (атомарність, узгодженість, ізольованість, довговічність) реляційні бази гарантують захист від втрати чи пошкодження даних навіть у разі збоїв, що особливо важливо для фінансових розрахунків, обліку балансу коштів, логування транзакцій і ведення історії змін.

Вибір між MongoDB та SQL-базою залежить від характеру та динаміки даних, що обробляються платформою. Якщо очікується робота з великою кількістю різнорідних чи мінливих структур (наприклад, інвентарі користувачів, додаткові атрибути лотів, кастомні налаштування), MongoDB надасть вищу гнучкість та масштабованість. Якщо ж основним є забезпечення суворої структурованості, складних зв'язків між даними, гарантованої цілісності транзакцій — безумовною перевагою буде реляційна база даних.

Для багатьох сучасних проєктів оптимальним підходом є використання гібридної архітектури: наприклад, PostgreSQL для фінансових розрахунків і критичних транзакцій, а MongoDB — для зберігання гнучких профілів користувачів, інвентарів, логів чи аналітики. Такий підхід дозволяє одночасно скористатися перевагами обох технологій та мінімізувати їхні обмеження.

Узагальнюючи, основні критерії вибору бази даних для трейдингової платформи виглядають так:

– якщо пріоритет — гнучкість структури, швидке масштабування та простота роботи з неструктурованими даними, оптимальним вибором стане MongoDB;

– якщо ключове значення мають транзакційна цілісність, сувора логіка зв’язків між таблицями та гарантія збереження кожної фінансової операції, краще обирати реляційну SQL-базу (PostgreSQL, MySQL);

– для складних систем із різними типами даних та високими вимогами до надійності доцільно поєднувати обидва підходи — кожен для свого підмодуля платформи.

2.4 Інтеграція зі Steam API та іншими сервісами

Для трейдингової платформи віртуальних товарів інтеграція зі Steam API та суміжними зовнішніми сервісами є ключовою умовою повноцінної функціональності, масштабованості та конкурентоспроможності. Steam, як найбільша цифрова екосистема для відеоігор і ринку ігрових предметів, надає розробникам потужний набір API для взаємодії з акаунтами, інвентарем користувачів, інформацією про ігри, а також для організації торгівлі у напівавтоматичному або автоматизованому режимі.

Реалізація такої інтеграції починається з отримання доступу до Steam Web API, що забезпечує автентифікацію користувачів через OpenID, дозволяє запитувати публічну інформацію про профілі, отримувати дані про ігрові інвентарі, транзакції, історію обміну тощо. Ключовою перевагою цієї інтеграції є можливість синхронізувати стан інвентаря у режимі реального часу: користувачі бачать свої актуальні предмети, а платформа може перевіряти їхній статус, унікальні характеристики, рідкість або цінність для подальших операцій [13] [18].

Важливим аспектом є автоматизація трейдингу через Steam Bot API — спеціальні програмні модулі, що здійснюють обмін предметами, надсилання й прийняття трейд-офферів без прямої участі людини. Це суттєво прискорює процеси купівлі-продажу, знижує кількість помилок, мінімізує затримки при масових операціях і дає змогу будувати складні сценарії для маркетплейсів, наприклад, групові обміни, P2P-платформи, гнучке управління резервами. З практичного погляду, правильно реалізований бот може обробляти тисячі транзакцій щодня, автоматично перевіряти автентичність лотів, обмеження, статус акаунтів та відповідність предметів умовам платформи [13] [18].

Крім Steam API, сучасні трейдингові платформи активно інтегруються з іншими сервісами — наприклад, платіжними шлюзами (PayPal, Stripe, криптовалютні гаманці), аналітичними системами (для відстеження трендів і цін), сторонніми маркетплейсами, месенджерами (Discord, Telegram) для сповіщень та автоматизації підтримки користувачів. Впровадження відкритих RESTful або GraphQL API відкриває можливість взаємодії з партнерами, зовнішніми інструментами аналітики, мобільними додатками чи професійними трейдерськими платформами [13] [18].

Окремої уваги потребує питання безпеки та відповідності політикам Steam. Оскільки Steam суворо регламентує правила використання API та автоматизованих ботів, платформа повинна дотримуватися принципів етичного використання даних, не допускати перевищення лімітів, захищати токени доступу та приватну інформацію користувачів, регулярно оновлювати протоколи безпеки. Недотримання цих вимог може призвести до блокування ключів API, тимчасової або постійної втрати доступу, а у деяких випадках навіть до юридичних наслідків [13] [18].

Важливим викликом є й питання стабільності інтеграції: Steam API та сторонні сервіси можуть зазнавати змін у форматах відповіді, тимчасових збоїв, оновлень політик доступу чи капчі. Для забезпечення безперервної роботи платформи необхідно передбачати обробку помилок, автоматичне повторення запитів,

резервування токенів, а також гнучке логування для швидкого виявлення і усунення проблем [13] [18].

Узагальнюючи, інтеграція зі Steam API та іншими сервісами для сучасної трейдингової платформи охоплює:

– автентифікацію та авторизацію користувачів через Steam OpenID та OAuth;

– синхронізацію інвентарів, отримання даних про предмети та історію транзакцій;

– реалізацію Steam-ботів для автоматизації трейдингу та обміну лотами;

– підключення платіжних шлюзів, аналітичних платформ і месенджерів для розширення функціоналу;

– постійний моніторинг безпеки, контроль лімітів, оновлення політик і гнучке логування для підтримки стабільності інтеграції.

2.5 Інструменти для CI/CD та розгортання (Docker, GitLab CI, Kubernetes)

Для забезпечення безперервної інтеграції та доставки використаємо стандартний CI/CD-pipeline. Наприклад, GitLab CI/CD чи GitHub Actions може автоматично збирати проект, запускати тести і деплоїти на сервер при кожному пуші в гілку main. Це гарантує швидкий фідбек і стабільність релізів.

З боку інфраструктури популярний підхід – контейнеризація з Docker. Backend і frontend будуть запаковані як Docker-образи, що спрощує управління залежностями. Для продакшн-розгортання плануємо використовувати Kubernetes (або аналог – кластер контейнерів). Як показав приклад проєкту EZX, система в Docker оркеструється Kubernetes для забезпечення гнучкості масштабування. Kubernetes автоматично створює нові репліки сервісу при зростанні навантаження і підтримує відмовостійкість (перезапускає впавші контейнери).

В тестових середовищах можна застосувати Docker Compose для локального розгортання всіх сервісів (Node.js, MongoDB, Redis тощо) у складі одного проекту. Для сповіщень та управління логами інтегруємо сервіси як Sentry (моніторинг помилок) і Prometheus/Grafana (моніторинг продуктивності).

Таким чином, за допомогою Docker/Kubernetes і CI/CD ми отримаємо швидке і надійне розгортання, а також можливість масштабувати платформи відповідно до зростання аудиторії. Приклад описаного стеку можна знайти в кейсі розробки платформи EZX: там застосунок працює в Docker-контейнерах на GCP, а Kubernetes керує масштабом сервісів.

Висновки до розділу 2

Другий розділ присвячений аналізу й вибору сучасних технологій та інструментів для розробки трейдингового вебзастосунку. Розглянуто найпопулярніші фреймворки фронтенду, серверні технології, бази даних, особливості інтеграції зі Steam API, а також засоби CI/CD і автоматизації розгортання. У підсумку, обрано оптимальний стек технологій для реалізації продуктивного, масштабованого та безпечного рішення, де особливу роль відіграють Node.js, React, MongoDB та Docker/Kubernetes. Цей підхід дає змогу створювати сучасний вебзастосунок, що відповідає вимогам ринку, легко масштабується й забезпечує надійність навіть за високих навантажень.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ДЛЯ ТРЕЙТИНГУ ВІРТУАЛЬНИХ ТОВАРІВ З ПЛАТФОРМИ STEAM

3.1 Архітектурна схема застосунку

Архітектура є тим наріжним каменем, на якому будується успіх сучасного вебзастосунку у сфері трейдингу віртуальними товарами Steam. Її продумана структура визначає не тільки технічні можливості платформи, а й рівень довіри користувачів, здатність витримувати пікові навантаження, реагувати на загрози, адаптуватися до змін ринку й ефективно взаємодіяти з такими потужними зовнішніми екосистемами, як Steam. Тому ще на початковому етапі проектування було прийнято рішення впровадити багаторівневу, модульну та інтеграційну архітектуру, яка об’єднує сучасні технології, дозволяє ефективно розподіляти ресурси, масштабувати рішення та забезпечувати високу відмовостійкість.

Ключовою ідеєю є побудова системи навколо концепції чіткої сегментації логічних рівнів із максимальним розмежуванням відповідальності. У даному випадку архітектура включає чотири основних структурні рівні: клієнтський, серверний, рівень зберігання даних та інтеграційний рівень зовнішніх API. Це дає змогу не тільки спростити розробку та підтримку окремих частин застосунку, але й гнучко реагувати на виклики, що виникають під час експлуатації. Докладно розглянемо кожний із цих рівнів, підкреслюючи їхню роль у реалізації поставлених бізнес-завдань і технічних вимог.

Почнемо з клієнтського рівня. Тут використовується підхід “Single Page Application” на базі React. Така модель дозволяє побудувати максимально інтерактивний, динамічний інтерфейс, що реагує на події без повного перезавантаження сторінки. Користувач отримує можливість швидко авторизуватись через Steam OpenID, переглядати власний ігровий інвентар, відправляти торгові запити, оперативно отримувати сповіщення про зміну статусу своїх угод у реальному

часі. Фронтенд спілкується із серверною частиною через HTTP(S)-запити та канал WebSocket, який дозволяє досягати мінімальних затримок при передачі подій, критичних для трейдингових операцій.

Серверний рівень розроблено на основі Node.js з використанням фреймворка Express. Цей рівень є “мозковим центром” застосунку, де реалізовано основну бізнес-логіку, маршрутизацію, обробку запитів, перевірку прав доступу, взаємодію з базою даних і Steam API. Тут також впроваджуються захисні механізми, такі як валідація вхідних даних, аутентифікація за допомогою JWT-токенів, обмеження частоти запитів (rate limiting) і аудит ключових дій користувача. Завдяки асинхронній моделі Node.js сервер ефективно працює під час одночасного обслуговування сотень і тисяч паралельних сесій, що особливо важливо для маркетплейсу.

Зберігання даних організовано із залученням гнучкої, масштабованої бази MongoDB, що дозволяє легко працювати з документами різної структури, зберігати історію змін, інвентар, профілі користувачів і записи про трейди. Для завдань, де потрібна сувора транзакційна цілісність (наприклад, фінансові розрахунки або міжкористувацькі розрахунки), можна використовувати реляційну базу PostgreSQL, однак для переважної більшості CRUD-операцій MongoDB ідеально підходить, дозволяючи оптимізувати продуктивність платформи. Додатково застосовується Redis як інструмент кешування часто запитуваних даних (наприклад, фрагментів інвентарю або публічної інформації про трейди), що мінімізує затримки й навантаження на основну базу.

Особливу увагу приділено інтеграції зі Steam API. Це не лише питання отримання списку предметів користувача чи створення трейд-пропозиції, а комплексна задача забезпечення стабільності, захищеності й відповідності частим оновленням зовнішнього сервісу.

Система повинна стійко обробляти помилки, пов’язані з тимчасовою недоступністю Steam, обмеженнями частоти запитів (rate limits), змінами в

протоколах. Усі чутливі ключі та токени зберігаються у спеціальних секціях конфігурації, що не потрапляють до публічного репозиторію, та захищаються через механізми оточення або Docker secrets. Система сповіщень у реальному часі базується на WebSocket і дозволяє користувачам моментально дізнаватися про зміни в трейдах, появу нових пропозицій, закриття угод чи виникнення системних подій. Усе це значно підвищує рівень довіри до платформи й формує відчуття “живої”, інтерактивної системи. Для автоматичного моніторингу стану застосунку, логування помилок і аномалій інтегруються сервіси Sentry та Grafana — це важливо не лише для розробників, а й для оперативного вирішення критичних ситуацій під час живої експлуатації.

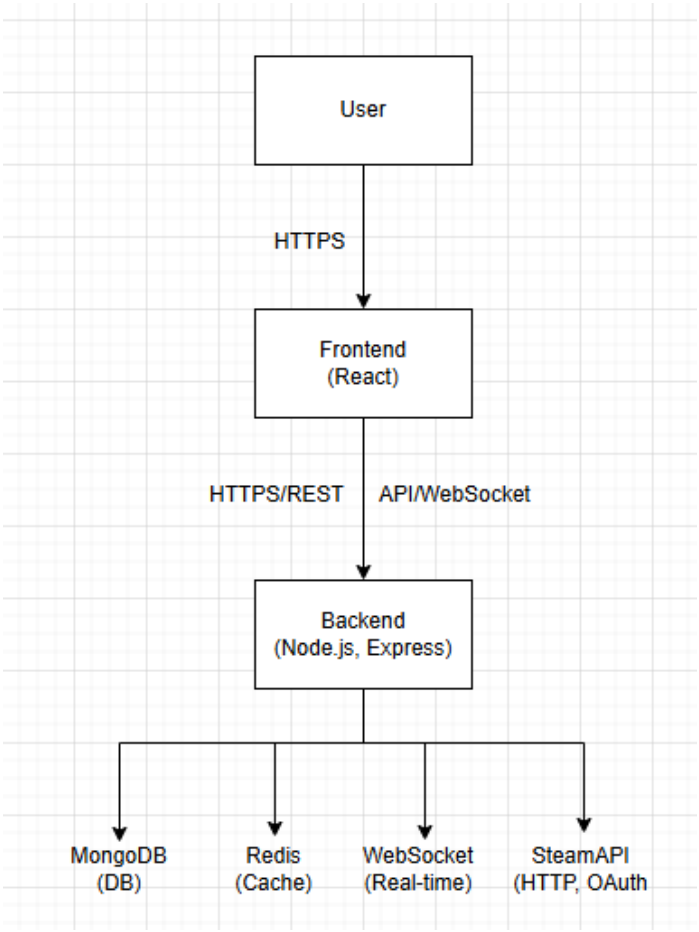

Рисунок 3.1 – Логічна архітектура вебзастосунку для трейдингу

На схемі (рис. 3.1) можна побачити, як користувач через зручний інтерфейс взаємодіє з серверною частиною, яка, у свою чергу, працює з базою даних, кешем, WebSocket-сервером та зовнішнім Steam API. Таке розмежування дозволяє уникнути «вузьких місць», швидко масштабувати будь-який із компонентів, а також ізолювати можливі проблеми для зручної діагностики й швидкого усунення.

Таблиця 3.1 – Розподіл відповідностей між компонентами архітектури

Компонент	Основна функція	Технології
Front-end	UI/UX, аутентифікація, WebSocket, інтеграція з API	React, WebSocket
Back-end (API)	Бізнес-логіка, інтеграція зі Steam, робота з базою	Node.js, Express
База даних	Зберігання користувачів, інвентарю, історії, трейдів	MongoDB
Кешування	Прискорення доступу до популярних даних	Redis
Steam API	Інвентар, трейди, автентифікація через OpenID	Steam Web API, OAuth
Моніторинг	Логування, аудит, оповіщення про збої	Sentry, Grafana

Ще одним ключовим елементом архітектури є взаємодія з користувачем у реальному часі. Зокрема, WebSocket-сервер дозволяє організувати двосторонній зв’язок між фронтендом і бекендом, що забезпечує оперативне оновлення статусів,

повідомлень і подій у системі. Такі підходи значно підвищують відчуття “живої” платформи та сприяють підвищенню лояльності користувачів.

Модель даних у платформі будується з урахуванням основних бізнес-сутностей: користувачі (з унікальними SteamID), предмети інвентарю (з урахуванням різних ігор Steam), трейди, історія операцій, рейтинги та відгуки. Гнучка структура MongoDB дозволяє органічно розширювати модель без необхідності міграцій та перерв у роботі застосунку. З метою забезпечення швидкодії в системі використовуються індекси за ключовими полями (SteamID, tradeID, дата створення тощо), що прискорює пошук і дозволяє працювати з великими обсягами даних навіть у реальних умовах навантаження.

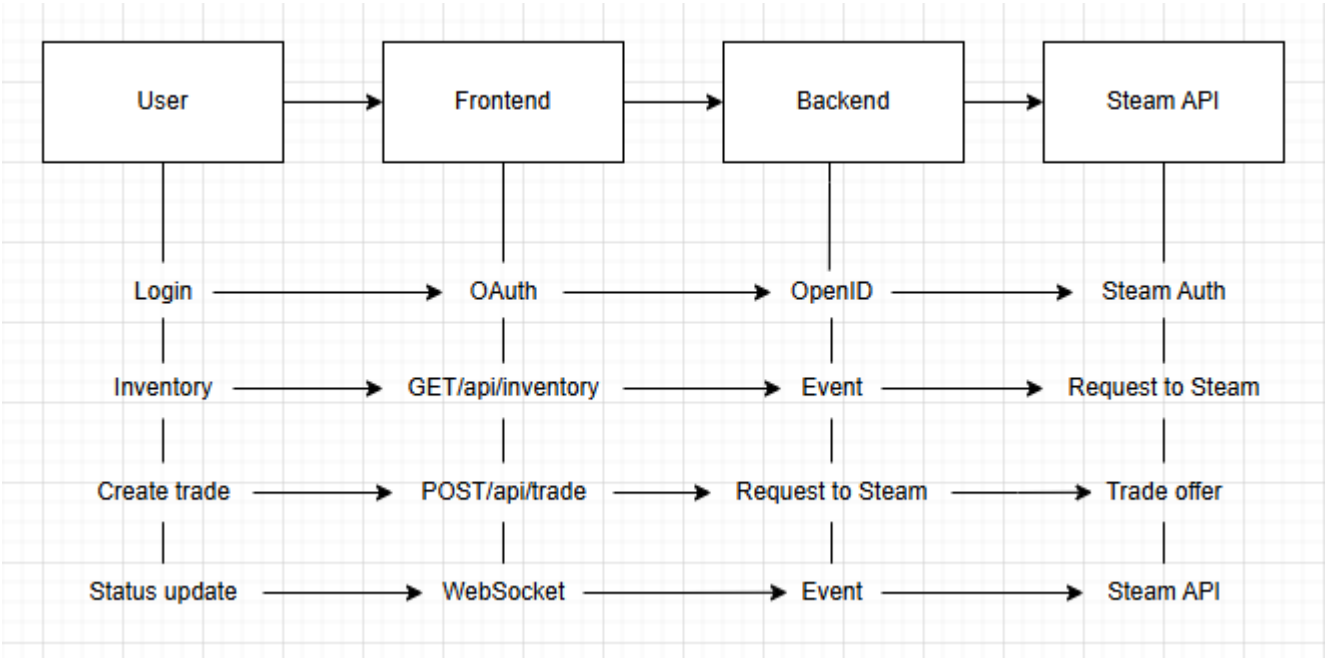

Рисунок 3.2 – Схема послідовності подій при створенні трейду

Особливістю даної архітектури є також наявність модулів захисту й безпеки, які інтегруються на кожному етапі. Для авторизації використовуються лише перевірені зовнішні провайдери (Steam OpenID), всі чутливі дані шифруються під час передачі,

API-запити захищені від класичних атак через валідацію, токенізацію і впровадження CORS-політики. Зберігання токенів та ключів відбувається у захищеному середовищі, а додатковий рівень контролю забезпечують системи логування та моніторингу.

З огляду на вищезазначене, вибір такої архітектурної моделі цілком виправданий. Вона забезпечує:

– високу швидкодію завдяки асинхронній обробці подій та кешування;

– надійну інтеграцію із зовнішніми сервісами, такими як Steam;

– відкритість до розширення: за потреби можна легко додати модулі без зміни основи;

– простоту масштабування — розгортання нових екземплярів компонентів відбувається без зупинки системи;

– високий рівень безпеки та довіри користувачів.

3.2 Функціональні модулі системи

Побудова ефективного й масштабованого вебзастосунку для трейдингу віртуальними товарами Steam вимагає чіткого розділення системи на функціональні модулі. Такий підхід дозволяє забезпечити гнучкість у розробці, тестуванні та підтримці, а також зменшує ризики під час масштабування чи розширення платформи. Модульність системи сприяє її стійкості до змін у зовнішніх API, змін у вимогах бізнесу чи навіть появі нових конкурентних сценаріїв використання. Кожен функціональний модуль виконує свою чітко окреслену роль і взаємодіє з іншими модулями через чітко визначені інтерфейси, що значно спрощує інтеграцію та модернізацію системи.

В основі модульної структури даної платформи лежить ідея, що кожна ключова функція має бути інкапсульована в окремий програмний компонент. Це дозволяє уникнути дублювання логіки, підвищує безпеку, сприяє зменшенню кількості багів, а також робить можливим незалежний розвиток різних частин системи. Крім того,

завдяки такому підходу платформа може легко масштабуватися шляхом горизонтального розгортання критичних модулів, а також забезпечувати гнучкість при впровадженні нових функцій чи зміні існуючих.

Основні модулі та їх призначення. Усі функціональні можливості платформи можна умовно поділити на кілька логічних блоків, кожен із яких має свій набір задач, відповідальність та інтеграційні точки. Серед ключових модулів системи варто виділити:

– *модуль аутентифікації та авторизації* — відповідає за безпечний вхід користувачів у систему, інтеграцію з Steam OpenID, створення нових профілів і управління сесіями. Саме цей модуль забезпечує ідентифікацію користувачів, перевірку їхніх прав доступу, а також контроль за “життєвим циклом” сесії. У ньому реалізовані механізми захисту від несанкціонованого доступу, мультифакторної аутентифікації, управління токенами (JWT) і контроль терміну дії сесій;

– *модуль інвентарю* — дозволяє користувачу переглядати, синхронізувати й фільтрувати власний інвентар у Steam. Модуль забезпечує отримання актуальних даних через Steam API, кешування для пришвидшення відгуку, відображення характеристик предметів (наприклад, рідкість, стан, гра), а також синхронізацію при кожному вході чи зміні складу інвентарю. Окремим завданням є оптимізація частоти звернень до Steam API для уникнення блокування через перевищення лімітів;

– *модуль трейдингу* — ключовий компонент платформи, який дає змогу ініціювати, приймати, відхиляти та завершувати трейди між користувачами. Тут реалізується основна бізнес-логіка: перевірка валідності пропозиції, резервування предметів, обробка змін статусів трейду, а також інтеграція з Steam TradeOffer API для проведення фактичних обмінів. Також у цьому модулі впроваджено логування всіх дій користувача з трейдами, аудит операцій та обробку винятків у разі невдачі (наприклад, відхилення трейду через відсутність предмету або зміну статусу профілю у Steam);

– *модуль історії операцій* — відповідає за зберігання повного логу дій користувача: створення й прийом трейдів, зміни інвентарю, отримання або відправку предметів. Завдяки цьому модулю користувачі мають змогу відслідковувати свою активність, а адміністрація — проводити аудит і розслідування у випадку виникнення спірних ситуацій;

– *модуль рейтингів і довіри* — формує рейтингову систему учасників платформи. На основі кількості й якості завершених трейдів, кількості успішних угод, відгуків інших користувачів, система автоматично нараховує рейтингові бали, що впливають на довіру до конкретного акаунта. Це особливо важливо для запобігання шахрайству й підвищення якості торгівлі;

– *модуль сповіщень у реальному часі* — організовує механізм миттєвого інформування користувачів про зміни у статусі трейдів, появу нових пропозицій, системні повідомлення та критичні помилки. Здійснюється на основі WebSocket, що гарантує мінімальні затримки та найвищу якість досвіду користувача;

– *модуль адміністративного управління* — надає адміністраторам інструменти для контролю за діяльністю користувачів, модерації трейдів, аналізу аномалій, розгляду скарг та впровадження політик безпеки. Тут також формується звітність, відбувається перегляд журналів і налаштування системних параметрів.

Описані модулі взаємодіють між собою через визначені API та подієві механізми. Особливо важливим для сучасного трейдингового майданчика є гнучкість структури: система повинна легко адаптуватися до появи нових бізнес-вимог, регуляторних норм, інтеграції з додатковими сервісами чи навіть впровадження нових типів торгівлі. Саме модульна модель дозволяє змінювати або оновлювати функціональні блоки незалежно один від одного, без ризику для роботи основної системи.

Таблиця 3.2 – Ключові функціональні модулі трейдингової платформи

Модуль	Основна відповідальність	Основні інтеграції
Аутентифікація/авторизація	Вхід через Steam, управління токенами	Steam OpenID, JWT
Інвентар	Синхронізація та відображення предметів	Steam API, MongoDB/Redis
Трейдинг	Створення/обробка трейдів, бізнес-логіка обміну	Steam TradeOffer API, DB
Історія операцій	Зберігання всіх дій та транзакцій	MongoDB, WebSocket
Рейтинги й довіра	Обробка балів, аналіз відгуків	Внутрішній API, DB
Сповіщення	Real-time повідомлення користувачу	WebSocket
Адміністративне управління	Модерація, аудит, політики, звітність	DB, внутрішній API

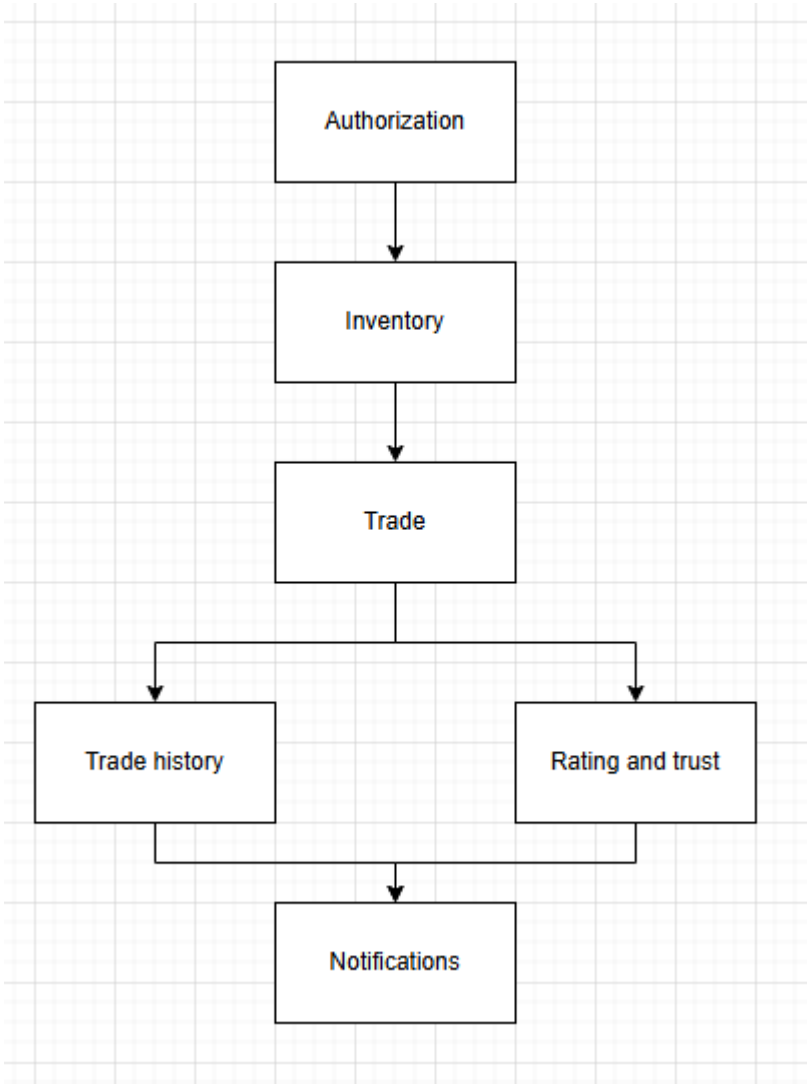

Рисунок 3.3 – Схема взаємозв’язків функціональних модулів трейдингової системи

Важливим аспектом розробки модульної системи є її тестованість. Оскільки кожен функціональний блок ізольований, це дозволяє проводити юніт-тестування окремих частин, а також легко впроваджувати автоматизовані end-to-end сценарії для перевірки інтеграції. Такий підхід суттєво спрощує виявлення й локалізацію помилок, підвищує швидкість розвитку та загальну надійність платформи.

3.4 Структура бази даних

У класичних вебсистемах давно домінували реляційні (SQL) СУБД, які чудово підходять для бізнесів із суворо фіксованою структурою даних, чіткими транзакціями, малим числом різновидів об’єктів. Проте сфера геймінгу й онлайн-трейдингу — це світ динаміки, різнорідних предметів, часто оновлюваних інвентарів, нових типів угод, бонусів, тимчасових івентів і навіть неочікуваних “аномалій” у структурі даних. Саме тому платформи, подібні до нашої, потребують максимальної гнучкості й здатності швидко еволюціонувати разом із ринком. MongoDB ідеально вписується у цей контекст, бо:

– підтримує вкладені документи будь-якої складності та довжини — інвентар гравця з різних ігор, бонуси, скіни, властивості предметів (навіть якщо завтра Steam додасть новий тип речей — схема не “зламається”);

– забезпечує надзвичайно швидку роботу з масивами об’єктів (наприклад, обробка тисячі предметів в інвентарі гравця, запити по угодах або історії);

– дозволяє горизонтально масштабуватися (шардінг), зберігаючи стабільну швидкодію навіть при зростанні до сотень тисяч користувачів;

– гарантує просту інтеграцію з Node.js/Express та сучасними front-end фреймворками;

– надає можливість гнучко індексувати поля, оптимізуючи пошук і фільтрацію по найбільш популярних сценаріях (наприклад, трейди певного SteamID або предмети за типом/грою).

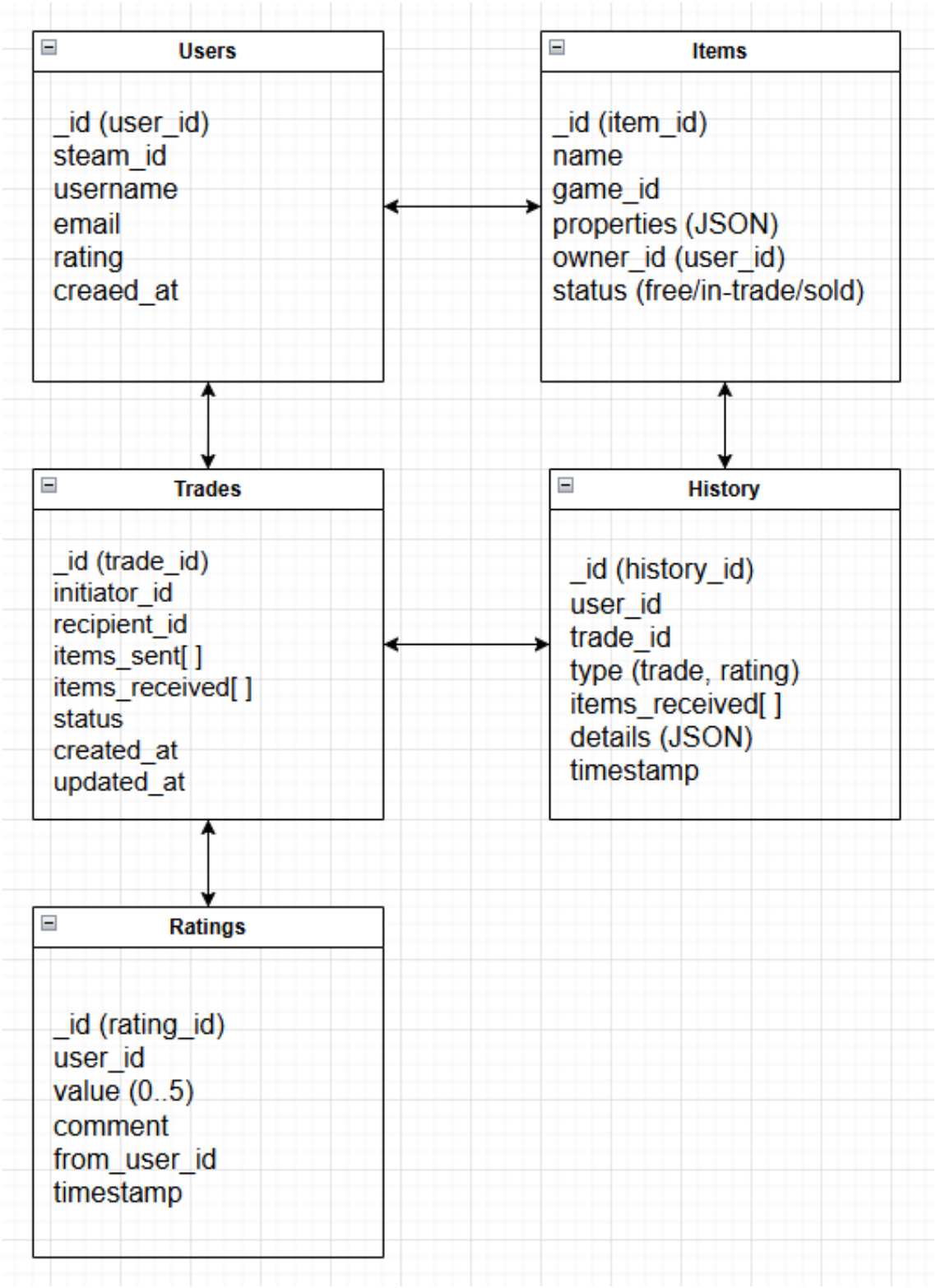

Рисунок 3.4 - Логічна модель бази даних трейдингового вебзастосунку

Структура, яка зображена на рисунку 3.4 дозволяє:

– зберігати інвентарі будь-якої складності (будь-яка кількість предметів, ігор, атрибутів);

– масово оновлювати чи синхронізувати інвентарі з Steam API без ризику для інших колекцій;

– забезпечити зв’язок трейдів із учасниками та предметами через посилання (id);

– легко агрегувати статистику по трейдах, користувачах, типах операцій через потужний aggregation pipeline.

Типові зв’язки тут реалізуються через references (наприклад, поле owner_id в Item містить _id користувача з колекції Users, а поле trade_id в History — посилання на документ Trades).

Документ користувача:

```
{
  "_id": "u17x5g2",
  "steam_id": "7656119XXXXXX",
  "username": "Vitazzyk",
  "email": "user@email.com",
  "rating": 4.82,
  "created_at": "2024-05-30T16:14:22Z"
}
```

Документ предмета інвентарю:

```
{
  "_id": "itm00452",
  "name": "AWP | Dragon Lore",
  "game_id": "cs2",
  "properties": {
```

```
        "rarity": "Red",
        "condition": "Factory New",
        "skin_id": "123456"
      },
      "owner_id": "u17x5g2",
      "status": "in-trade",
      "image_url":              "https://cdn.steam.com/item/img/awp-
dragonlore.png"
    }
```

Документ трейду:

```
    {
      "_id": "tr5432",
      "initiator_id": "u17x5g2",
      "recipient_id": "u21m0k7",
      "items_sent": [
        { "item_id": "itm00452", "game_id": "cs2" }
      ],
      "items_received": [
        { "item_id": "itm00452", "game_id": "cs2" }
      ],
      "status": "pending",
      "created_at": "2024-05-30T16:20:01Z",
      "updated_at": null
    }
```

Документ історії подій:

```
    {
      "_id": "hst9042",
      "user_id": "u17x5g2",
      "trade_id": "tr5432",
```

```
      "type": "trade_create",
      "details": {
        "message": "Користувач ініціював трейд із користувачем u21m0k7"
      },
      "timestamp": "2024-05-30T16:20:01Z"
    }
```

Документ рейтингу:

```
    {
      "_id": "r8743",
      "user_id": "u21m0k7",
      "from_user_id": "u17x5g2",
      "value": 5,
      "comment": "Угода пройшла ідеально, рекомендую!",
      "timestamp": "2024-05-30T17:11:00Z"
    }
```

Для максимальної швидкодії у MongoDB використовують індекси по найбільш затребуваних полях. Зокрема:

– steam_id — для швидкого пошуку користувача за ідентифікатором Steam;

– owner_id у Items — для швидкого вибірки інвентарю певного користувача;

– status у Trades — для масової фільтрації по актуальних чи завершених трейдах;

– timestamp у History — для ефективної побудови хронології подій;

– комбіновані індекси (наприклад, user_id + trade_id у History) — для швидкої агрегації операцій по конкретних сценаріях.

Таблиця 3.3 – Приклади основних запитів у MongoDB для трейдингу

Запит	Колекція	Опис/результат
Знайти всі предмети користувача	Items	{ owner_id: "u17x5g2" }
Всі трейди зі статусом “pending”	Trades	{ status: "pending" }
Всі дії користувача в історії	History	{ user_id: "u17x5g2" }
Середній рейтинг користувача	Ratings	Aggregation по user_id і value
Останні 10 трейдів певного SteamID	Trades	{ $or: [{ initiator_id: "..."} , {recipient_id: "..."}] } з сортуванням

3.5 Архітектура та реалізація інтелектуального чат-бота для пошуку скінів у Steam

Сучасні торгові платформи, особливо у сфері геймінгу та цифрових предметів, вимагають інноваційних рішень для взаємодії з користувачем. Одним із таких рішень є впровадження інтелектуального чат-бота, який, будучи інтегрованим прямо у вебдодаток (на окремій сторінці), дозволяє користувачеві шукати товари за допомогою природної мови, без необхідності вручну налаштовувати фільтри чи знати точні назви предметів. У цьому розділі детально розглядається архітектура, принципи роботи й особливості розробки такого чат-бота для пошуку скінів у Steam із використанням технологій машинного навчання.

Ідея чат-бота полягає в тому, щоб зробити пошук і підбір віртуальних товарів максимально простим, швидким та інтуїтивно зрозумілим для користувача. Замість

традиційної навігації по розгалужених каталогах, відбору за десятками чекбоксів, користувач просто відкриває окрему сторінку “Чат-бот”, де у вікні діалогу пише, наприклад: “хочу собі скин на AK-47 чорного кольору”, “дай щось дешеве для USP-S”, “шукаю красиву AWP не дорожче 500 гривень”. Система, завдяки інтегрованому модулю машинного навчання (NLP), самостійно розпізнає, що саме шукає користувач, інтерпретує його побажання (зброя, колір, бюджет, стан, стиль тощо) й повертає найбільш релевантні результати — список скінів із цінами, фото, короткими описами та посиланням на покупку.

Ключем до цього є саме використання сучасних моделей обробки природної мови (Natural Language Processing). Чат-бот не просто “ловить” окремі слова — він розуміє контекст, синоніми, відтінки значень, навіть неформальні чи геймерські сленгові запити. Це досягається завдяки попередньо навчальним моделям на кшталт BERT, DistilBERT чи аналогічних embedding-моделей. Коли користувач надсилає повідомлення, текст потрапляє у серверний ML-модуль, де проходить попередню обробку (очищення, токенізація, виявлення ключових сутностей: назва зброї, ознака кольору, ціновий діапазон, стиль тощо). Далі запит кодується у векторну форму, і по такій же схемі закодовано описи товарів у базі даних (MongoDB чи Elasticsearch із підтримкою semantic search). Порівнюючи вектори (через cosine similarity чи інший metric), бот обирає ті скіни, які максимально “близькі” до побажань користувача, навіть якщо слова не збігаються буквально (“темний” = “чорний”, “дешевий” = “до 200 грн”, “у мінімальному зносі” = “Minimal Wear”).

Крім того, чат-бот підтримує повноцінний діалог: якщо початковий запит надто загальний або суперечливий (“хочу щось класне для AWP”), бот автоматично уточнить деталі — “Який стиль чи колір цікавить?”, “Який максимальний бюджет?”, “Який стан (Factory New, Minimal Wear тощо)?”. Це реалізується через систему сценаріїв та діалогових шаблонів: бот розпізнає “пробіли” у специфікації й довантажує уточнення, поки не отримає достатньо параметрів для релевантної видачі.

На сервері працює окремий модуль, який, отримавши структурований запит від NLP, звертається до колекції скінів у базі даних. Для цього для кожного скіна додатково зберігається не лише технічний опис, а й embedding-вектор, що дозволяє реалізовувати як класичний фільтр (по ціні, зброї, стану), так і векторний пошук по semantic близькості. Ось приклад того, як виглядає документ у базі.

```
    {
      "_id": "itm392",
      "weapon": "AK-47",
      "name": "Black Laminate",
      "color": "чорний",
      "price": 280,
      "wear": "Field-Tested",
      "description": "Classic black finish, solid laminate stock",
      "embedding": [0.112, -0.083, ...]
}
```

Коли бот отримує запит типу “шукаю темний скин для AK-47 до 300 гривень”, він спершу перевіряє фільтри (weapon = AK-47, price ≤ 300), далі ранжує результати за векторною близькістю до embedding-запиту користувача. Це гарантує, що навіть новачок, який не знає точних назв або не використовує стандартну термінологію, отримає релевантні варіанти.

Для швидкості видачі й масштабованості використовується або Elasticsearch (із підтримкою vector search), або MongoDB Atlas Search, які дозволяють зберігати й індексувати мільйони embedding-векторів, підтримувати миттєвий пошук навіть при пікових навантаженнях.

Чат-бот працює у реальному часі: користувач вводить запит, отримує відповідь майже миттєво. Якщо серед знайденого — жоден варіант не задовольняє, бот може

запропонувати розширити бюджет, вибрати іншу комбінацію стану/кольору, а також автоматично показати “схожі” скіни (на базі embedding-подібності).

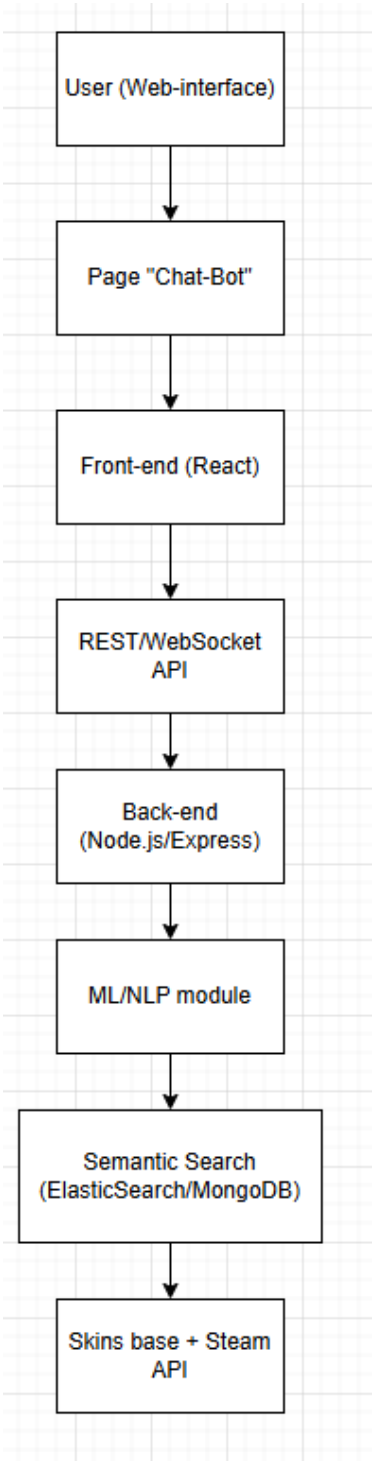

Рисунок 3.5 – Архітектура чат-бота в структурі додатку

Окрему увагу у розробці чат-бота приділено тестуванню й безпеці. Кожна функція, від обробки вхідного тексту до ранжування результатів, покривається unit та integration тестами (Jest, Mocha, Cypress). Вся взаємодія з базою — через захищені API, без прямих ін’єкцій. Забезпечено фільтрацію некоректних/токсичних запитів, обмеження на “спам” чи нецензурну лексику, валідацію кожного запиту на стороні серверу.

Масштабування досягається завдяки горизонтальному розгортанню окремих ML-сервісів (наприклад, через Docker Swarm чи Kubernetes), а також кешуванню популярних embedding-запитів і результатів.

Інтеграція із Steam API дозволяє підтримувати завжди актуальні дані про асортимент, ціни, наявність скінів, їх статуси. Автоматичні скрипти регулярно оновлюють цю інформацію у базі, щоб жоден користувач не отримав застарілих чи неактуальних пропозицій. Це підвищує довіру до платформи і дозволяє оперативно реагувати на зміни на ринку.

Реалізація чат-бота напряму у вебдодатку дозволяє не тільки залучити нову аудиторію, але й суттєво підвищити конверсію: користувач, що прийшов “просто подивитися”, швидше знаходить бажаний скин, отримує позитивний досвід і повертається до платформи. Бот легко інтегрується у будь-який SPA, не вимагає додаткового програмного забезпечення від користувача, його інтерфейс максимально простий і знайомий кожному, хто користувався месенджерами чи сервісами підтримки.

Приклад реальної взаємодії з ботом:

– користувач: “Потрібен дешевий Glock із зеленим кольором.”;

– бот: “Ось що я знайшов: Glock-18 | Bunsen Burner (зелений, 115₴), Glock-18 | Moonrise (зелено-чорний, 130₴). Хочете фільтрувати за станом?”;

– користувач: “Мінімальний знос.”;

– бот: “Супер! Glock-18 | Moonrise | Minimal Wear (135₴), Glock-18 | Sacrifice | Minimal Wear (140₴). Додати до обраного?”.

Такий сценарій підкреслює основну перевагу: користувач просто пише, як говорить у житті, а система робить усю “чорнову” роботу.

Впровадження інтелектуального чат-бота на сторінці додатку — це якісно новий етап у розвитку цифрових торгових платформ. Поєднання NLP-моделі, гнучкої архітектури пошуку, інтеграції з актуальними даними Steam та UX, орієнтованого на людину, забезпечує платформі конкурентну перевагу, глибоку персоналізацію та високий рівень довіри користувачів. Такий підхід не лише відповідає трендам світового e-commerce, а й створює умови для впровадження smart-рекомендацій, автоматизації підтримки та навіть нових форматів маркетингу у майбутньому.

3.6 Візуалізація та користувацький інтерфейс вебдодатку

Якість користувацького інтерфейсу (UI) та зручність взаємодії (UX) є критично важливими факторами успіху сучасного вебдодатку, особливо у сфері трейдингу цифровими товарами, де рішення про покупку чи взаємодію часто приймається емоційно і за лічені секунди. В цьому розділі розглянуто структуру, логіку й особливості візуального оформлення платформи для трейдингу скінами зі Steam — від концепції головної сторінки до унікального розділу з інтелектуальним чат-ботом.

Весь інтерфейс побудовано з урахуванням сучасних тенденцій: мінімалізм, адаптивність до різних пристроїв, прості форми, яскраві акценти для важливих дій, максимальна “читабельність” і швидкість доступу до основних функцій. Кольорова схема витримана у темно-сірих та синіх тонах із яскравими інтерактивними елементами (наприклад, кнопки «Придбати», «Додати у вибране», тощо), що органічно вписується у геймерську естетику Steam-ринку.

Головна сторінка одразу занурює користувача у атмосферу трейдингу: великий банер із коротким описом можливостей платформи, кнопки швидкої навігації

(наприклад, «Переглянути скіни», «Спробувати чат-бот»), відображення топових або нових товарів із інтерактивними картками. Кожна картка містить фото скіна, його назву, ціну, рейтинг, короткий опис та кнопку для перегляду детальнішої інформації або додавання у обране.

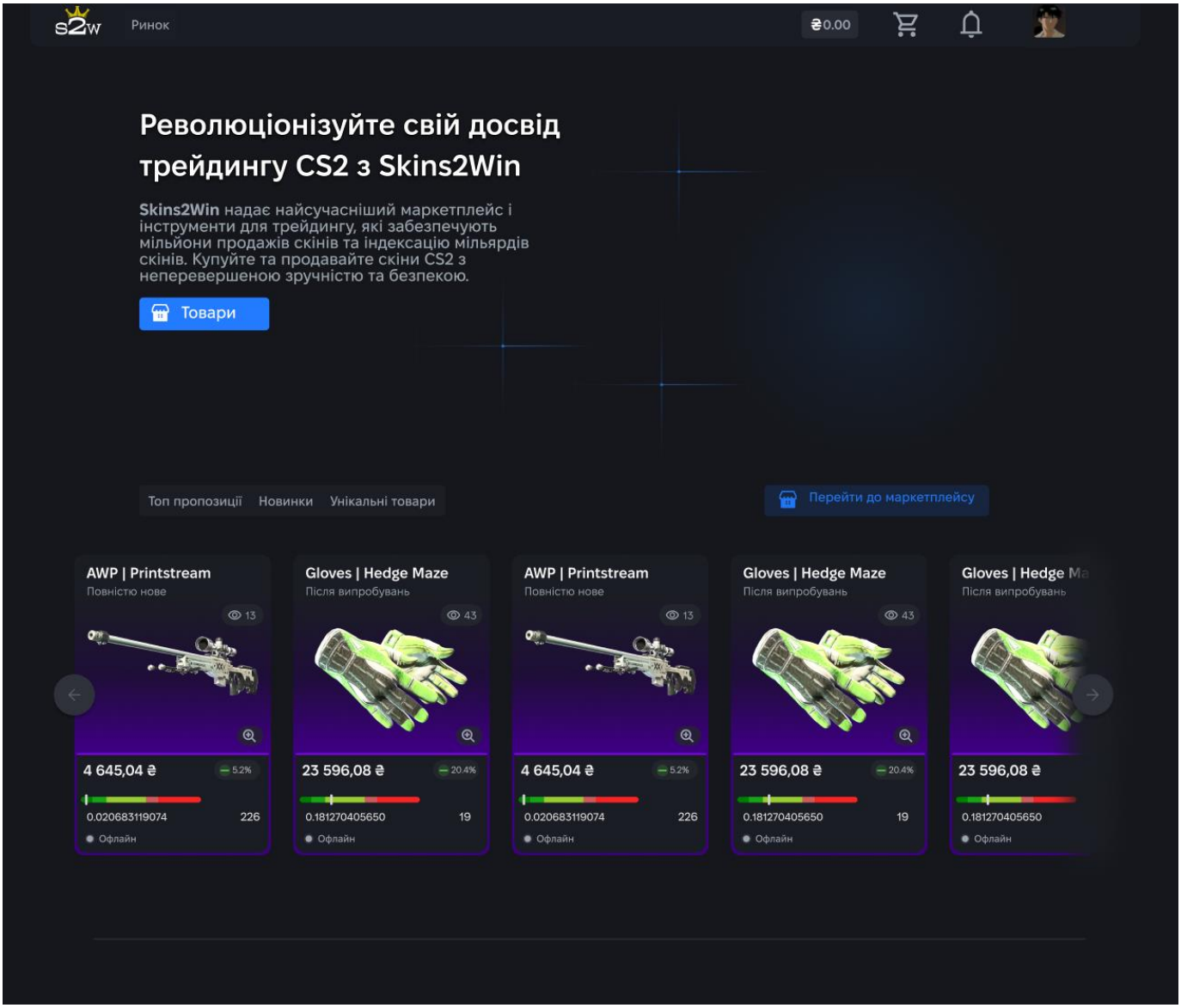

Рисунок 3.6 – Головне меню вебдодатку

Рисунок 3.7 – Картка скіну

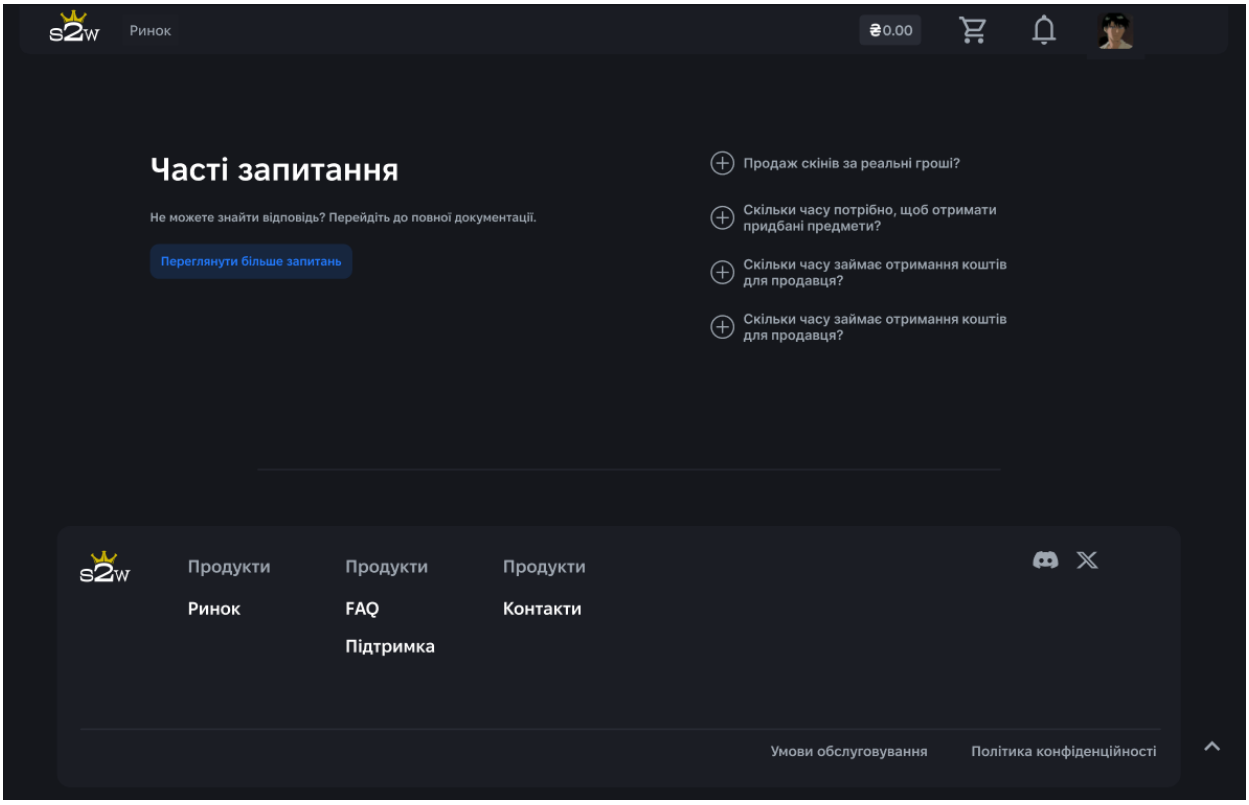

Рисунок 3.8 – Сторінка з відповіддю на часті запитання

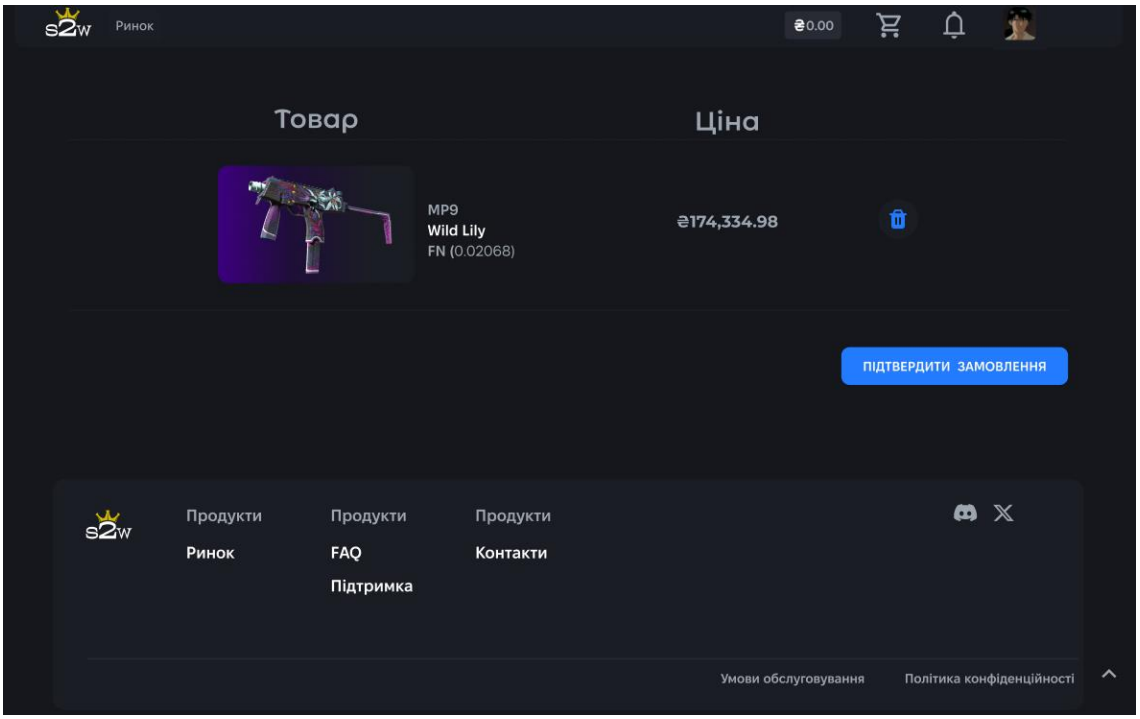

Рисунок 3.9 – Підтвердження замовлення

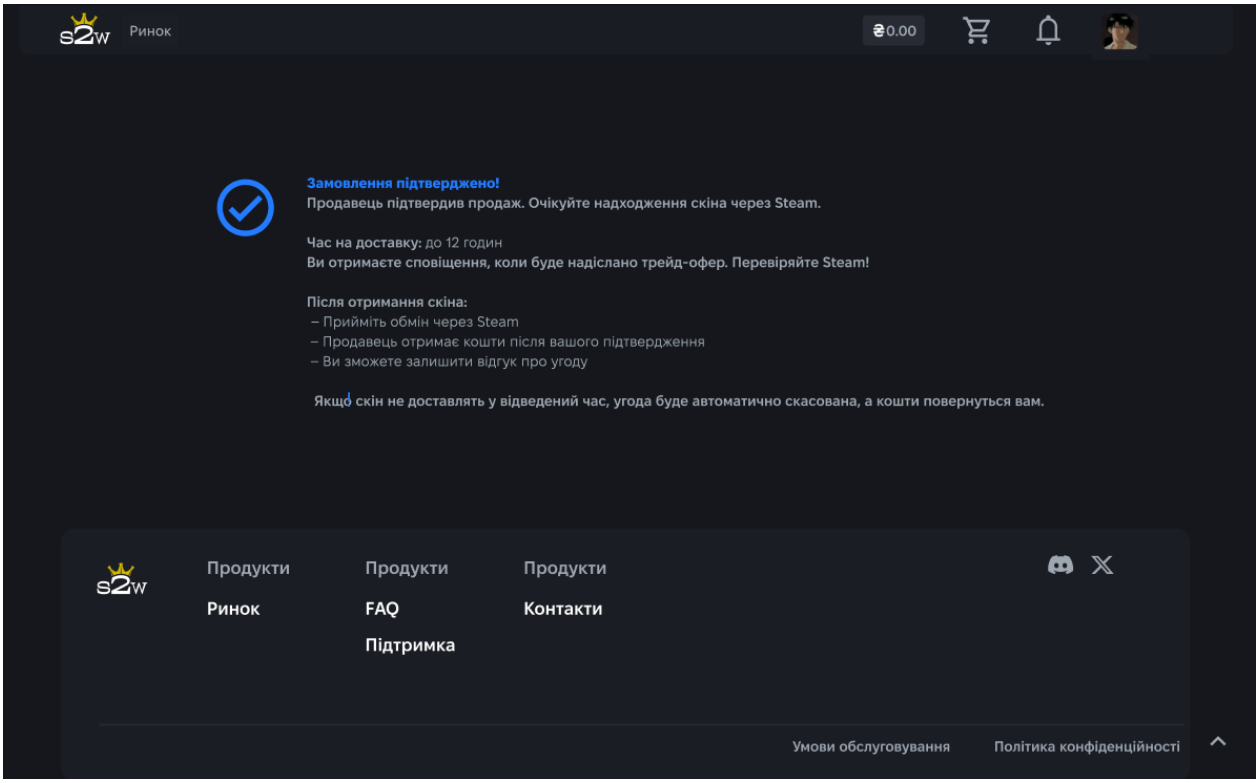

Рисунок 3.10 – Отримання замовлення

Меню навігації закріплено у верхній частині екрану — воно дозволяє перейти на основні сторінки: «Ринок», «Мої трейди», «Чат-бот», «Профіль», а для адміністратора — у розділ керування товарами. Меню максимально лаконічне, не перевантажене, з інтуїтивно зрозумілими іконками.

Окрема увага приділена сторінці інтелектуального чат-бота. Вона відкривається у вигляді знайомого всім месенджер-інтерфейсу: праворуч — поле для діалогу, де відображаються як повідомлення користувача, так і відповіді бота у вигляді інтерактивних блоків із пропозиціями товарів (фото, опис, кнопка “Купити”), а ліворуч — коротка інструкція чи підказки щодо формування запитів. Інтерфейс максимально чистий: нічого не відволікає від головної функції — пошуку й вибору скінів через природний діалог.

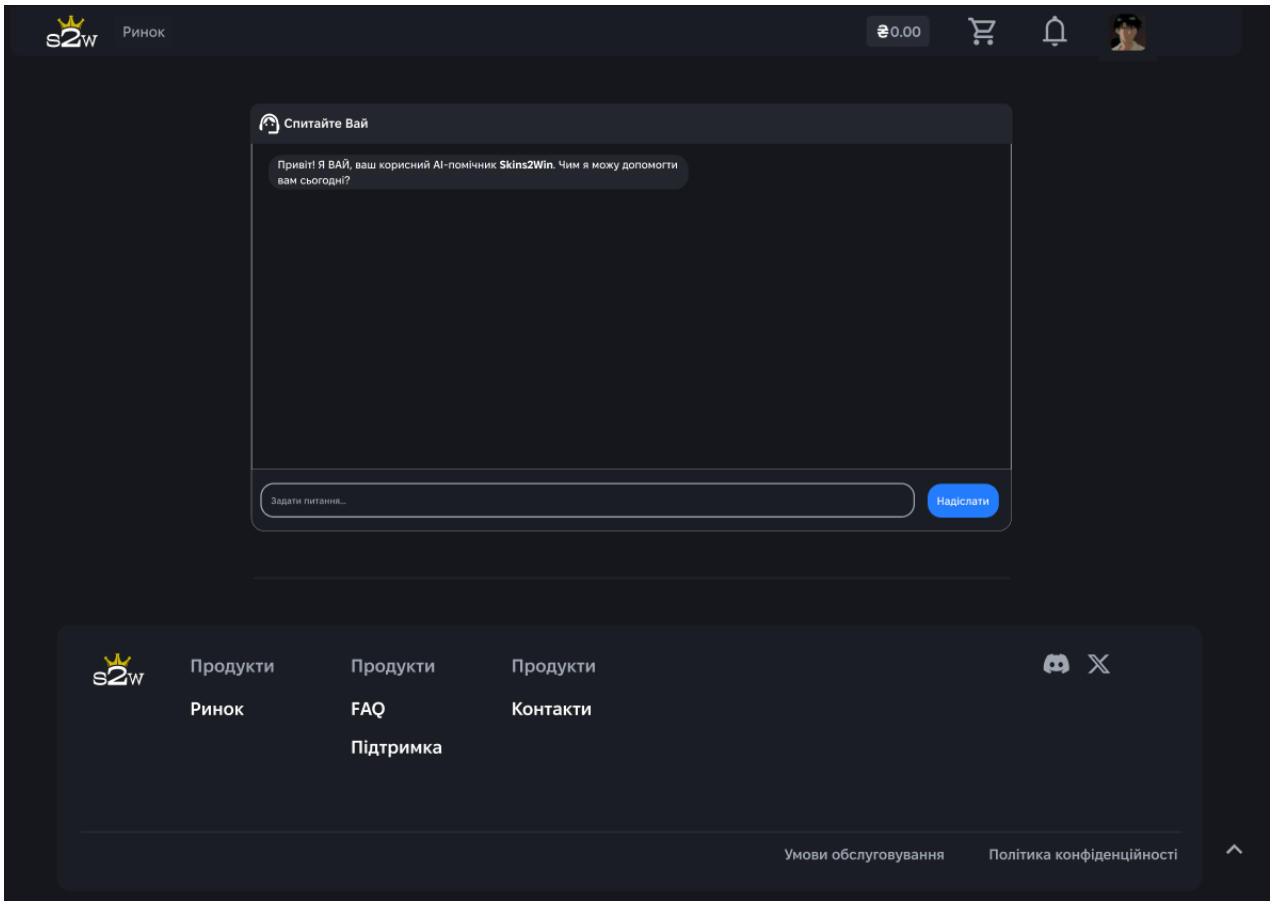

Рисунок 3.11 – Чат-бот вебзастосунку

При відкритті товару на окремій сторінці користувач бачить велике зображення скіна, детальні характеристики (назва, гра, рідкість, стан, ціна, наявність, рейтинг), а нижче — пропозиції схожих товарів або коментарі покупців. Усе оформлено у вигляді адаптивної “картки”, яка добре виглядає і на десктопі, і на мобільному пристрої.

Особливу увагу приділено “реал-тайм” елементам: наприклад, коли користувач ініціює трейд чи додає товар до обраного, відповідна кнопка анімується, статус оновлюється без перезавантаження сторінки, а у випадку чат-бота нові відповіді підвантажуються у діалог миттєво, імітуючи живу розмову.

Адаптивність — ще одна важлива характеристика. Всі сторінки, елементи й блоки розроблено за принципом mobile-first, із плавним масштабуванням і перебудовою сітки для смартфонів, планшетів, широких моніторів. Це забезпечує комфортну роботу навіть для гравців, які заходять з мобільного під час ігрової сесії.

Для покращення сприйняття нових користувачів реалізовано інтерактивний “онбординг”: при першому вході платформа коротко знайомить з основними можливостями, показує підказки щодо роботи із пошуком, чат-ботом, налаштуваннями профілю та безпеки.

Висновки до розділу 3

У третьому розділі розкрито особливості проєктування та реалізації власного вебзастосунку для трейдингу віртуальних товарів Steam. Детально описано архітектуру системи, функціональні модулі, структуру бази даних, принципи інтеграції зі Steam API, а також впровадження інтелектуального чат-бота та побудову користувацького інтерфейсу. Описані рішення дозволяють досягнути гнучкості, надійності та високої продуктивності платформи, а також забезпечити позитивний користувацький досвід завдяки сучасному інтерфейсу й автоматизованим сервісам підтримки. У результаті, реалізований застосунок демонструє конкурентні переваги як з боку функціоналу, так і з точки зору безпеки й масштабованості.

4 ПЕРСПЕКТИВИ РОЗВИТКУ ТА МАЙБУТНІ ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ

4.1 Аналіз тенденцій розвитку ринку віртуальних товарів

Глобальні прогнози свідчать про стійке зростання ринку: за оцінками Mordor Intelligence, обсяг ринку віртуальних товарів зросте з 112.33 млрд USD (2025) до 261.36 млрд (2030) при середньому річному зростанні 18.4%. Причинами називають зростання онлайн-ігор, популяризацію метавсесвітів, та масову інтеграцію блокчейн/NFT-технологій. У 2023–2024 роках кількість геймерів перевищуватиме 3 млрд осіб, більшість із яких – активні споживачі внутрішньоігрових покупок.

Серед основних тенденцій:

– *розвиток метавсесвітів і AR/VR.* Інтеграція віртуальних товарів у доповнену/віртуальну реальність розширює потенціал ринку. Наприклад, цифровий одяг і аксесуари для аватарів у VR-іграх чи соціальних платформах;

– *насичення NFT-базованими моделями.* Провідні компанії гейміндустрії освоюють NFT: створюють токенізовані версії унікальних предметів, а також інвестують у віртуальні землі (Decentraland, Sandbox тощо) для отримання доходу від оренди чи перепродажу. Це підвищує довіру до цифрових активів завдяки прозорій реєстрації власності;

– *підвищений інтерес до аналітики.* Зростає попит на інструменти прогнозування цін і трендів. Аналітика великих даних може допомагати трейдерам приймати рішення. Наприклад, у 2024 р. DMarket представила алгоритми ML-прогнозування для ціни та попиту;

– *розваги ігрової індустрії.* Багатокористувацькі free-to-play проєкти (Fortnite, Apex Legends, LoL) стимулюють попит на косметичні скіни та аксесуари. Геймери усе частіше сприймають віртуальні товари як соціальні символи – бажання «виразити себе» в онлайні веде до зростання витрат на скіни.

4.2 Інтеграція з блокчейн- та DeFi-платформами

Однією з ключових перспектив є глибша інтеграція з блокчейном і DeFi. На відміну від традиційних майданчиків, що тримають записи в базах даних, децентралізовані рішення дають користувачам більший контроль над активами. NFT-маркетплейси (OpenSea, Rarible) вже показали попит на торгівлю цифровим мистецтвом. У майбутньому можна очікувати, що ігрові предмети будуть подібно токенізовані: кожен предмет матиме свій унікальний NFT, який підтверджує власність.

DMarket демонструє один з варіантів такого шляху: він використовує смарт-контракти Ethereum для забезпечення умов угоди і зберігання даних. Це гарантує, що навіть якщо сайт DMarket буде недоступним, дані про угоди залишаться в блокчейні. З появою нових рішення типу Layer-2 або інших блокчейнів (Binance Smart Chain, Polygon, Solana) може з’явитися інтеграція DeFi-функцій: наприклад, використання ігрових предметів як застави, видача позик під NFT, або обмін токенами екосистеми.

Крім того, криптовалюти вже застосовуються як засіб оплати при купівлі ігрової валюти чи скінів. Використання стейблкойнів усуває волатильність і прискорює розрахунки. Наприклад, DMarket і CSFloat підтримують біткоїн та USDT. У перспективі може реалізовуватися «універсальний гаманець» для гравця, який зберігатиме різні токени і миттєво конвертуватиме їх під час транзакцій.

4.3 Розширення аналітики: алгоритми прогнозування цін і трендів

Розвиток штучного інтелекту та машинного навчання дозволяє покращити аналітику ринку. Уже зараз деякі сервіси намагаються прогнозувати ціну предметів на основі історії торгів. Згаданий DMarket вбудував ML-алгоритми, що аналізують роки даних торгів і дають AI-прогноз щодо найкращої ціни для продажу предмета.

Така система автоматично оцінює оптимальну ціну для продажу з урахуванням обраного періоду – довший період дає вищу цільову ціну.

У майбутньому можливе впровадження більш складних моделей (нейронні мережі, часові ряди), що враховують зовнішні фактори (тенденції популярності гри, оновлення контенту, сезонні коливання). Для трейдера це означатиме наявність інструментів, які підказуватимуть, коли вигідно купувати чи продавати. Високу точність також демонструють системи рекомендацій товарів (див. наступний пункт).

4.4 Впровадження системи рекомендацій на основі машинного навчання

З розвитком сучасних торгових платформ для віртуальних товарів усе більшого значення набуває персоналізація досвіду користувачів. Одним із найефективніших інструментів підвищення лояльності, залученості та обсягів торгівлі є системи рекомендацій, побудовані на основі технологій машинного навчання. Вони дозволяють пропонувати кожному користувачеві індивідуально релевантні товари, лоти, акції або інші сервіси, що суттєво збільшує ймовірність успішної транзакції й покращує загальний користувацький досвід.

Впровадження системи рекомендацій розпочинається з глибокого аналізу історичних даних. Платформа збирає інформацію про поведінку користувачів: історію пошуків, покупки, перегляди лотів, оцінки, час активності, інтереси до певних категорій чи окремих ігор. Далі ці дані очищаються та структуруються, після чого на їхній основі створюється модель користувацьких профілів. Такий підхід дозволяє враховувати не лише прямі взаємодії з товарами, а й приховані вподобання або закономірності поведінки.

Найбільш поширеними підходами до створення системи рекомендацій є колаборативна фільтрація, контент-орієнтовані та гібридні моделі. Колаборативна фільтрація аналізує схожість між користувачами або товарами: наприклад, система пропонує лоти, які були цікавими для інших користувачів із подібними вподобаннями

чи історією покупок. Контент-орієнтований підхід ґрунтується на аналізі характеристик самих товарів — наприклад, скіни тієї ж гри, з подібною рідкістю або за тією ж ціною, що і попередні покупки користувача. Гібридна модель поєднує обидва підходи, що дозволяє отримувати більш точні й персоналізовані рекомендації навіть у разі обмеженої кількості даних.

На практиці впровадження системи рекомендацій вимагає налаштування регулярного збору та обробки даних у реальному часі, інтеграції з основною базою платформи, а також розгортання окремих сервісів для генерації, оновлення та доставки рекомендацій користувачам. Особливу роль відіграють питання продуктивності — система має оперативно обробляти великі масиви даних та оновлювати пропозиції відповідно до змін у поведінці користувача або ринкових тенденцій. З цією метою використовуються сучасні бібліотеки машинного навчання (наприклад, Scikit-learn, TensorFlow, PyTorch), інструменти для обробки потокових даних (Kafka, Apache Spark), а також хмарні сервіси для масштабування (AWS Sagemaker, Google AI Platform).

Важливим аспектом є й безпека: всі зібрані дані повинні оброблятися анонімно, із дотриманням стандартів захисту персональної інформації, а результати рекомендацій мають бути прозорими для користувача — наприклад, з поясненням, чому саме ця пропозиція є для нього актуальною.

Узагальнюючи, основні етапи впровадження системи рекомендацій на основі машинного навчання на платформі виглядають так:

– збір і структуризація історичних даних про активність користувачів;

– розробка й навчання моделей машинного навчання (колаборативних, контент-орієнтованих чи гібридних);

– інтеграція рекомендаційної системи з основною платформою й розгортання API для швидкої доставки пропозицій;

– регулярне оновлення моделей відповідно до змін у поведінці користувачів і ринкових тенденцій;

– забезпечення прозорості, безпеки та дотримання конфіденційності даних.

Впровадження такої системи дозволяє не лише суттєво підвищити обсяги продажів, а й створити для користувачів унікальний досвід взаємодії з платформою, що є ключовим чинником у сучасній конкурентній боротьбі за увагу гравців на ринку віртуальних товарів.

4.5 Мультиплатформеність: мобільні пристрої та десктоп

Згідно з даними StatCounter, у квітні 2025 року близько 62,06% глобального веб-трафіку припадало на мобільні пристрої. Це означає, що вебзастосунок для трейдингу повинен бути доступний не лише на десктопі: він мусить коректно працювати у мобільних браузерах або мати окремий мобільний інтерфейс (PWA) чи навіть нативні додатки (на React Native чи Flutter).

У сучасних реаліях ринок вебзастосунків для трейдингу віртуальними товарами дедалі більше орієнтується на мультиплатформеність. Користувачі очікують, що можуть взаємодіяти із сервісом не лише з персонального комп’ютера, а й із мобільного пристрою — у будь-який час і в будь-якому місці. Саме тому питання підтримки як десктопної, так і мобільної версії стає стратегічним для кожної конкурентної платформи, оскільки безперервний доступ до ключових функцій і зручний інтерфейс суттєво впливають на залученість та лояльність аудиторії.

Створення мультиплатформеного рішення передбачає низку технічних і дизайнерських викликів. Необхідно забезпечити адаптивний і консистентний інтерфейс, який буде однаково зручним як на великому екрані десктопа, так і на невеликому мобільному пристрої. Для цього сучасні команди розробників активно застосовують принципи responsive design — динамічне налаштування верстки, елементів управління та відображення контенту під різні розміри екрана та типи

пристроїв. Важливою є і оптимізація завантаження: на мобільних пристроях обмежена пропускна здатність мережі та ресурси процесора, тому особливу увагу приділяють мінімізації обсягу даних, використанню lazy loading та кешуванню.

Ще одним підходом до досягнення мультиплатформеності є використання прогресивних вебзастосунків (PWA), які поєднують можливості звичайного сайту та мобільного додатку. PWA дозволяють користувачам встановлювати платформу як додаток на смартфон, отримувати push-сповіщення, працювати офлайн та забезпечують майже нативний користувацький досвід без необхідності завантажувати окремий додаток з App Store чи Google Play. Для повноцінної мобільної присутності платформи також можуть розробляти окремі нативні додатки, особливо якщо є потреба в глибокій інтеграції з функціями пристрою — наприклад, для швидких сповіщень, біометричної автентифікації чи роботи з камерою.

З боку бекенду і API важливо забезпечити однакову доступність усіх функцій для різних типів клієнтів — як мобільних, так і десктопних. Це означає уніфікований підхід до реалізації авторизації, управління лотами, сповіщеннями, фінансовими операціями та історією дій користувача. Особливу увагу слід приділяти безпеці мобільних додатків: шифрування даних, захист від MITM-атак, безпечне зберігання токенів і дотримання вимог до захисту персональної інформації.

Мультиплатформеність також стосується інтеграції із зовнішніми сервісами — наприклад, Steam API, платіжними шлюзами, аналітичними платформами. Всі ці інтеграції повинні бути доступними і для десктопної, і для мобільної версії, щоб користувач отримував повний спектр сервісів незалежно від пристрою.

Висновки до розділу 4

Четвертий розділ спрямований на оцінку перспектив подальшого розвитку трейдингових платформ та впровадження нових функціональних можливостей. Проаналізовано актуальні тенденції, такі як інтеграція з блокчейн- і DeFi-

технологіями, розвиток аналітичних інструментів для прогнозування цін, застосування систем рекомендацій на основі машинного навчання, а також мультиплатформеність (мобільні і десктопні додатки). Підкреслено, що впровадження інноваційних технологій є запорукою збереження конкурентоспроможності та підвищення цінності платформи для користувачів, а постійний розвиток та адаптація до ринкових трендів дозволяють забезпечити стабільне зростання й актуальність сервісу у майбутньому.

ВИСНОВКИ

Результати проведеної роботи свідчать про те, що розроблений вебзастосунок для трейдингу віртуальних товарів зі Steam є сучасним і гнучким рішенням, яке відповідає більшості вимог до функціоналу, безпеки та продуктивності на ринку. Разом із тим, під час дослідження й практичної реалізації були виявлені певні недоліки, які можуть стати об’єктом для подальшого вдосконалення. Зокрема, інтеграція зі Steam API накладає технічні обмеження, пов’язані з лімітами, нестабільністю відповіді сервісу та політиками самої платформи, що періодично ускладнює підтримку безперервної роботи трейдингу. Також залишається актуальним питання боротьби із шахрайством і забезпечення максимальної прозорості для користувачів, оскільки навіть найсучасніші технічні засоби не гарантують абсолютної безпеки в умовах постійного розвитку фішингових схем і соціальної інженерії.

Проєкт має значний потенціал для подальшого розвитку. Серед ключових перспектив — впровадження систем прогнозування цін і ринкових трендів за допомогою алгоритмів машинного навчання, розширення функціоналу для мобільних пристроїв, інтеграція з блокчейн- та DeFi-платформами для прозорих і швидких транзакцій, а також реалізація мультиплатформеного інтерфейсу для охоплення ширшої аудиторії користувачів. Особливу цінність для користувачів може мати розвиток системи рекомендацій та персоналізованої аналітики, що дозволить робити більш виважені рішення при купівлі чи продажу віртуальних предметів.

Окремо варто відзначити впровадження в систему інтелектуального чат-помічника, який став важливою складовою якісного користувацького досвіду. Завдяки застосуванню сучасних методів обробки природної мови та інтеграції з базою даних, чат-бот забезпечує швидкі консультації, допомогу в пошуку потрібних товарів, автоматизовані відповіді на часті запитання й підтримку в процесі здійснення операцій. Це дозволяє значно підвищити рівень сервісу, скоротити навантаження на

службу підтримки й зробити платформу більш зручною та інклюзивною для різних категорій користувачів.

Реалізований вебзастосунок демонструє високий рівень конкурентоспроможності й має широкі перспективи для вдосконалення та впровадження нових інновацій у сфері трейдингу цифровими товарами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Valve Corporation. Steam Community Market [Електронний ресурс]. – Режим доступу: https://steamcommunity.com/market/. – Дата звернення: 22.04.2025.

2. CSFloat. CSFloat – Marketplace for CS2 Skins [Електронний ресурс]. – Режим доступу: https://csfloat.com/. – Дата звернення: 23.04.2025.

3. Wikipedia. DMarket [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/DMarket. – Дата звернення: 22.04.2025.

4. CS.MONEY. Умови користування [Електронний ресурс]. – Режим доступу: https://cs.money/terms/. – Дата звернення: 25.04.2025.

5. DMarket Blog. Sales History [Електронний ресурс]. – Режим доступу: https://dmarket.com/blog/sales-history/. – Дата звернення: 26.04.2025.

6. Forbes Україна. Як DMarket змінив ринок віртуальних товарів [Електронний ресурс]. – Режим доступу: https://forbes.ua/technology/yak-dmarket-zminiv-rinok-virtualnih-tovariv-21042021-12345. – Дата звернення: 27.04.2025.

7. Site24x7 Blog. Scaling Node.js Applications [Електронний ресурс]. – Режим доступу: https://www.site24x7.com/blog/scaling-nodejs-applications/. – Дата звернення: 28.04.2025.

8. DMarket Blog. TrustShield [Електронний ресурс]. – Режим доступу: https://dmarket.com/blog/trustshield/. – Дата звернення: 29.04.2025.

9. Cleveroad Blog. Blockchain in Gaming: DMarket [Електронний ресурс]. – Режим доступу: https://cleveroad.com/blog/blockchain-in-gaming-dmarket/. – Дата звернення: 30.04.2025.

10. Integrate.io Blog. MongoDB vs MySQL: A Performance Comparison [Електронний ресурс]. – Режим доступу: https://integrate.io/blog/mongodb-vs-mysql/. – Дата звернення: 24.04.2025.

11. React Documentation. React. FAQ Internals [Електронний ресурс]. – Режим доступу: https://uk.reactjs.org/docs/faq-internals.html. – Дата звернення: 01.05.2025.

12. Axios. Axios Documentation [Електронний ресурс]. – Режим доступу: https://axios-http.com/docs/intro. – Дата звернення: 02.05.2025.

13. Valve Corporation. Steamworks Documentation. Web Browser Authentication with OpenID [Електронний ресурс]. – Режим доступу: https://partner.steamgames.com/doc/features/auth. – Дата звернення: 01.05.2025.

14. Fortune Business Insights. Virtual Goods Market Report 2023–2032 [Електронний ресурс]. – Режим доступу: https://www.fortunebusinessinsights.com/industry-reports/virtual-goods-market-113289. – Дата звернення: 25.04.2025.

15. Mordor Intelligence. Virtual Goods Market Size & Share Analysis [Електронний ресурс]. – Режим доступу: https://www.mordorintelligence.com/industry-reports/virtual-goods-market. – Дата звернення: 26.04.2025.

16. StatCounter Global Stats. Platform Market Share Worldwide [Електронний ресурс]. – Режим доступу: https://gs.statcounter.com/platform-market-share. – Дата звернення: 22.04.2025.

17. Гаврилюк О. В. Безпека вебзастосунків: сучасні підходи та рішення. // Інформаційні технології в освіті, науці та техніці. – 2023. – №2. – С. 39–44. – Режим доступу: https://itechjournal.edu.ua/articles/2023/02/gavryliuk.pdf (дата звернення: 21.04.2025).

18. Steam Web API Documentation. – Режим доступу: https://partner.steamgames.com/doc/webapi_overview (дата звернення: 23.04.2025).

19. Mozilla Developer Network. Introduction to Web Security. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Security (дата звернення: 24.04.2025).

20. Шумський І. Порівняння реляційних і NoSQL баз даних для учасних вебплатформ // Вісник КНУ. Серія: Інформатика. – 2022. – №8. – С. 75–83. – Режим

доступу: http://journal.knu.ua/informatics/2022/08/shumskyi.pdf (дата звернення: 26.04.2025).

21. Ant Design: UI framework for React. – Режим доступу: https://ant.design/docs/react/introduce (дата звернення: 27.04.2025).

22. OAuth 2.0 and OpenID Connect. – Режим доступу: https://oauth.net/2/ (дата звернення: 28.04.2025).

23. Docker Documentation: Get Started. – Режим доступу: https://docs.docker.com/get-started/ (дата звернення: 29.04.2025).

24. Kubernetes Documentation. – Режим доступу: https://kubernetes.io/docs/home/ (дата звернення: 01.05.2025).

25. Mongoose ODM for MongoDB. – Режим доступу: https://mongoosejs.com/docs/ (дата звернення: 04.05.2025).

26. Stripe API Documentation. – Режим доступу: https://stripe.com/docs/api (дата звернення: 06.05.2025).

27. Червінський А. Ю., Литвиненко Д. В. Штучний інтелект у рекомендаційних системах: досвід платформ e-commerce // Штучний інтелект. – 2023. – №1. – С. 55–62. – Режим доступу: https://ai-journal.edu.ua/2023/01/chervinskyi.pdf (дата звернення: 08.05.2025).

28. Prometheus Monitoring. – Режим доступу: https://prometheus.io/docs/introduction/overview/ (дата звернення: 12.05.2025).

29. Grafana Documentation. – Режим доступу: https://grafana.com/docs/grafana/latest/ (дата звернення: 13.05.2025).

30. Ярошенко П. М. Гейміфікація онлайн-сервісів: тенденції та кейси застосування // Інформаційне суспільство. – 2024. – №2. – С. 48–54. – Режим доступу: https://infosociety-journal.org/2024/02/yaroshenko.pdf (дата звернення: 15.05.2025).

31. Технології захисту API: рекомендації OWASP. – Режим доступу: https://owasp.org/www-project-api-security/ (дата звернення: 19.05.2025).

32. Jenkins CI Documentation. – Режим доступу: https://www.jenkins.io/doc/ (дата звернення: 24.05.2025).

33. Blockchain for Digital Assets: DMarket Case Study. – Режим доступу: https://dmarket.com/blog/blockchain-for-digital-goods/ (дата звернення: 28.05.2025).

ДОДАТОК А

Програмна реалізація

```
const express = require('express');
const session = require('express-session');
const passport = require('passport');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const http = require('http');
const { Server: SocketIOServer } = require('socket.io');
const path = require('path');
dotenv.config();
const PORT = process.env.PORT || 5000;
const MONGO_URI = process.env.MONGO_URI ||
'mongodb://127.0.0.1:27017/steamtradeapp';
mongoose.connect(MONGO_URI, { useNewUrlParser: true,
useUnifiedTopology: true })
  .then(() => console.log('✅ Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err));

require('./models/User');
require('./models/Item');
require('./models/Trade');

require('./config/steamPassport');
const app = express();
app.use(express.json());
app.use(session({
  secret: 'keyboard cat',  // секрет сесії (можна винести в .env)
  resave: false,
  saveUninitialized: false
}));
app.use(passport.initialize());
app.use(passport.session());
const cors = require('cors');
app.use(cors({ origin: 'http://localhost:3000', credentials: true }));
app.use('/api/auth', require('./routes/auth'));
app.use('/api/inventory', require('./routes/inventory'));
app.use('/api/listings', require('./routes/listings'));
app.use('/api/trades', require('./routes/trades'));
app.use(express.static(path.join(__dirname, '..', 'frontend',
'build')));
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '..', 'frontend', 'build',
'index.html'));
});
```

```
const server = http.createServer(app);
const io = new SocketIOServer(server, {
  cors: { origin: 'http://localhost:3000', methods: ['GET','POST'] }
});
io.on('connection', (socket) => {
  console.log('Client connected via WebSocket');
  socket.on('support_message', (msg) => {
    console.log('User message to support bot:', msg);
    let reply;
    const lowerMsg = msg.toLowerCase();
    if (lowerMsg.includes('привіт') || lowerMsg.includes('hello')) {
      reply = 'Вітаємо! Чим можу допомогти?';
    } else if (lowerMsg.includes('trade') ||
lowerMsg.includes('трейд')) {
      reply = 'Для здійснення трейду, переконайтеся що ви увійшли через
Steam і виставили предмет на продаж або вибрали предмет для покупки.';
    } else if (lowerMsg.includes('refund') ||
lowerMsg.includes('повернення')) {
      reply = 'Якщо продавець не надішле предмет протягом 72 годин,
угоду буде скасовано, а кошти повернуто покупцю.';
    } else {
      reply = 'Я – AI-помічник. Ваше питання передано до служби
підтримки. Очікуйте відповіді або перевірте FAQ.';
    }
    socket.emit('support_reply', reply);
  });

  socket.on('disconnect', () => {
    console.log('Client disconnected from WebSocket');
  });
});
const steamBot = require('./steamBot');
steamBot.init(io);  // ініціалізуємо бота та передаємо йому io для
сповіщень

server.listen(PORT, () => {
  console.log(`🚀 Server listening on port ${PORT}`);
});

const passport = require('passport');
const SteamStrategy = require('passport-steam').Strategy;
const mongoose = require('mongoose');
const User = mongoose.model('User');

passport.serializeUser((user, done) => {
  done(null, user.steamId);
});
```

```
passport.deserializeUser(async (id, done) => {
  try {
    const user = await User.findOne({ steamId: id });
    done(null, user);
  } catch (err) {
    done(err, null);
  }
});

passport.use(new SteamStrategy({
    returnURL: `${process.env.STEAM_REALM}api/auth/steam/return`,
    realm: process.env.STEAM_REALM,
    apiKey: process.env.STEAM_API_KEY
  },
  async (identifier, profile, done) => {
    const steamId = profile.id;
    try {
      let user = await User.findOne({ steamId });
      if (!user) {
        user = new User({
          steamId: steamId,
          displayName: profile.displayName,
          avatar: profile.photos && profile.photos.length ?
profile.photos[2].value : '',
          profileUrl: profile._json.profileurl || '',
          balance: 1000,
          tradeCount: 0,
          reputation: 0
        });
        await user.save();
      }
      return done(null, user);
    } catch (err) {
      return done(err);
    }
  }
));

const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
  steamId: { type: String, required: true, unique: true },
  displayName: String,
  avatar: String,
  profileUrl: String,
  balance: { type: Number, default: 0 },
  tradeCount: { type: Number, default: 0 },
  reputation: { type: Number, default: 0 }
```

```
});
mongoose.model('User', UserSchema);

const mongoose = require('mongoose');
const ItemSchema = new mongoose.Schema({
  name: { type: String, required: true },
  image: String,
  exterior: String,
  float: Number,
  price: { type: Number, required: true },
  seller: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  buyer: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  status: { type: String, default: 'active' },
  assetId: String,
  createdAt: { type: Date, default: Date.now }
});
mongoose.model('Item', ItemSchema);

const mongoose = require('mongoose');
const TradeSchema = new mongoose.Schema({
  item: { type: mongoose.Schema.Types.ObjectId, ref: 'Item', required:
true },
  seller: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
required: true },
  buyer: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required:
true },
  price: { type: Number, required: true },
  status: { type: String, default: 'pending_seller' },
  createdAt: { type: Date, default: Date.now },
  completedAt: Date,
  sellerRating: Number,
  buyerRating: Number
});
mongoose.model('Trade', TradeSchema);

const express = require('express');
const passport = require('passport');
const router = express.Router();

router.get('/steam', passport.authenticate('steam'), (req, res) => {});

router.get('/steam/return',
  passport.authenticate('steam', { failureRedirect: '/' }),
  (req, res) => {
    const jwt = require('jsonwebtoken');
```

```
        const token = jwt.sign({ steamId: req.user.steamId },
process.env.JWT_SECRET, { expiresIn: '1h' });
        res.redirect(`${process.env.STEAM_REALM}?token=${token}`);
    });

    router.get('/logout', (req, res) => {
      req.logout(() => {
        req.session.destroy(() => {
          res.redirect('/');
        });
      });
    });

    module.exports = router;

    const express = require('express');
    const axios = require('axios');
    const router = express.Router();

    function ensureAuth(req, res, next) {
      if (!req.user) {
        return res.status(401).json({ error: 'Not authenticated' });
      }
      next();
    }

    router.get('/', ensureAuth, async (req, res) => {
      const steamId = req.user.steamId;
      try {
        const url =
`https://steamcommunity.com/inventory/${steamId}/730/2?l=english&count=5000`
;
        const response = await axios.get(url);
        const data = response.data;
        if (!data || !data.descriptions) {
          return res.status(500).json({ error: 'Failed to fetch inventory
from Steam' });
        }
        const items = [];
        const descriptions = data.descriptions;
        const assets = data.assets;
        for (let asset of assets) {
          const descr = descriptions.find(d => d.classid === asset.classid
&& d.instanceid === asset.instanceid);
          if (!descr) continue;
          const item = {
            assetId: asset.assetid,
```

```
            name: descr.market_hash_name || descr.market_name ||
descr.name,
            image: descr.icon_url ? `https://steamcommunity-
a.akamaihd.net/economy/image/${descr.icon_url}` : '',
            tradable: descr.tradable,
            exterior: null
          };
          if (descr.tags) {
            const exteriorTag = descr.tags.find(t => t.category ===
"Exterior");
            if (exteriorTag) item.exterior = exteriorTag.name;
          }
          if (item.tradable === 0) continue;
          items.push(item);
        }
        res.json({ items });
      } catch (err) {
        console.error('Inventory fetch error:', err);
        res.status(500).json({ error: 'Error fetching inventory' });
      }
    });

    module.exports = router;

    const express = require('express');
    const mongoose = require('mongoose');
    const router = express.Router();
    const Item = mongoose.model('Item');
    const User = mongoose.model('User');

    router.get('/', async (req, res) => {
      try {
        const items = await Item.find({ status: 'active'
}).populate('seller', 'displayName tradeCount reputation');
        res.json(items);
      } catch (err) {
        console.error('Error fetching listings:', err);
        res.status(500).json({ error: 'Failed to fetch listings' });
      }
    });

    function ensureAuth(req, res, next) {
      if (!req.user) return res.status(401).json({ error: 'Not
authenticated' });
      next();
    }
```

```
router.post('/', ensureAuth, async (req, res) => {
  try {
    const { assetId, name, image, exterior, price } = req.body;
    if (!assetId || !name || !price) {
      return res.status(400).json({ error: 'Missing required fields'
});
    }
    let existing = await Item.findOne({ assetId: assetId, status:
'active' });
    if (existing) {
      return res.status(400).json({ error: 'Item already listed for
sale' });
    }
    const newItem = new Item({
      name,
      image,
      exterior,
      float: req.body.float || undefined,
      price,
      seller: req.user._id,
      assetId: assetId,
      status: 'active'
    });
    await newItem.save();
    res.status(201).json(newItem);
  } catch (err) {
    console.error('Error creating listing:', err);
    res.status(500).json({ error: 'Failed to create listing' });
  }
});

module.exports = router;

const express = require('express');
const mongoose = require('mongoose');
const router = express.Router();
const Trade = mongoose.model('Trade');
const Item = mongoose.model('Item');
const User = mongoose.model('User');

function ensureAuth(req, res, next) {
  if (!req.user) return res.status(401).json({ error: 'Not
authenticated' });
  next();
}

router.post('/buy/:itemId', ensureAuth, async (req, res) => {
```

```
        const buyer = req.user;
        const itemId = req.params.itemId;
        try {
          const item = await Item.findById(itemId).populate('seller');
          if (!item || item.status !== 'active') {
            return res.status(400).json({ error: 'Item not available' });
          }
          if (buyer.balance < item.price) {
            return res.status(400).json({ error: 'Insufficient balance to buy
this item' });
          }
          const trade = new Trade({
            item: item._id,
            seller: item.seller._id,
            buyer: buyer._id,
            price: item.price,
            status: 'pending_seller'
          });
          await trade.save();
          item.status = 'sold';
          item.buyer = buyer._id;
          await item.save();
          buyer.balance -= item.price;
          await buyer.save();
          if (req.app.get('io')) {
            req.app.get('io').to(item.seller.steamId).emit('trade_request', {
tradeId: trade._id, item: item.name });
          }
          res.status(201).json(trade);
        } catch (err) {
          console.error('Buy item error:', err);
          res.status(500).json({ error: 'Failed to create trade' });
        }
      });

      router.post('/:tradeId/confirm', ensureAuth, async (req, res) => {
        const tradeId = req.params.tradeId;
        try {
          const trade = await
Trade.findById(tradeId).populate('item').populate('seller').populate('buyer'
);
          if (!trade) return res.status(404).json({ error: 'Trade not found'
});
          if (trade.seller._id.toString() !== req.user._id.toString()) {
            return res.status(403).json({ error: 'Only the seller can confirm
this trade' });
          }
```

```
        if (trade.status !== 'pending_seller') {
          return res.status(400).json({ error: 'Trade is not awaiting
seller confirmation' });
        }
        trade.status = 'seller_confirmed';
        await trade.save();
        trade.status = 'buyer_received';
        await trade.save();
        if (req.app.get('io')) {
          req.app.get('io').to(trade.buyer.steamId).emit('trade_update', {
tradeId: trade._id, status: 'seller_confirmed' });
        }
        res.json({ success: true });
      } catch (err) {
        console.error('Seller confirm error:', err);
        res.status(500).json({ error: 'Failed to confirm trade' });
      }
    });

    router.post('/:tradeId/receive', ensureAuth, async (req, res) => {
      const tradeId = req.params.tradeId;
      try {
        const trade = await
Trade.findById(tradeId).populate('seller').populate('buyer');
        if (!trade) return res.status(404).json({ error: 'Trade not found'
});
        if (trade.buyer._id.toString() !== req.user._id.toString()) {
          return res.status(403).json({ error: 'Only the buyer can confirm
receipt' });
        }
        if (trade.status !== 'buyer_received' && trade.status !==
'seller_confirmed') {
          return res.status(400).json({ error: 'Trade is not awaiting buyer
confirmation' });
        }
        trade.status = 'completed';
        trade.completedAt = new Date();
        await trade.save();
        const seller = await User.findById(trade.seller._id);
        if (seller) {
          seller.balance += trade.price;
          seller.tradeCount += 1;
          seller.reputation += 1;
          await seller.save();
        }
        if (req.app.get('io')) {
```

```
            req.app.get('io').to(trade.seller.steamId).emit('trade_update', {
tradeId: trade._id, status: 'completed' });
            req.app.get('io').to(trade.buyer.steamId).emit('trade_update', {
tradeId: trade._id, status: 'completed' });
          }
          res.json({ success: true });
        } catch (err) {
          console.error('Buyer receive error:', err);
          res.status(500).json({ error: 'Failed to complete trade' });
        }
      });

      router.post('/:tradeId/rate', ensureAuth, async (req, res) => {
        const tradeId = req.params.tradeId;
        const { rating } = req.body;
        try {
          const trade = await Trade.findById(tradeId);
          if (!trade || trade.status !== 'completed') {
            return res.status(400).json({ error: 'Trade not found or not
completed' });
          }
          if (trade.seller.toString() === req.user._id.toString()) {
            trade.sellerRating = rating;
          } else if (trade.buyer.toString() === req.user._id.toString()) {
            trade.buyerRating = rating;
          } else {
            return res.status(403).json({ error: 'Not a participant of this
trade' });
          }
          await trade.save();
          res.json({ success: true });
        } catch (err) {
          console.error('Rating error:', err);
          res.status(500).json({ error: 'Failed to submit rating' });
        }
      });

      router.get('/my', ensureAuth, async (req, res) => {
        try {
          const trades = await Trade.find({ $or: [ { seller: req.user._id },
{ buyer: req.user._id } ] })
                                    .populate('item')
                                    .populate('seller', 'displayName')
                                    .populate('buyer', 'displayName');
          res.json(trades);
        } catch (err) {
          console.error('Fetch trade history error:', err);
```

```
      res.status(500).json({ error: 'Failed to fetch trade history' });
    }
  });

  module.exports = router;

  const SteamUser = require('steam-user');
  const SteamTotp = require('steam-totp');
  const TradeOfferManager = require('steam-tradeoffer-manager');
  const SteamCommunity = require('steamcommunity');

  const steamClient = new SteamUser();
  const steamCommunity = new SteamCommunity();
  const manager = new TradeOfferManager({
    steam: steamClient,
    community: steamCommunity,
    language: 'en'
  });

  const botName = process.env.BOT_USERNAME;
  const botPassword = process.env.BOT_PASSWORD;
  const sharedSecret = process.env.BOT_SHARED_SECRET;
  const identitySecret = process.env.BOT_IDENTITY_SECRET;

  function init(io) {
    const logOnOptions = {
      accountName: botName,
      password: botPassword
    };
    if (sharedSecret) {
      logOnOptions.twoFactorCode                                    =
SteamTotp.generateAuthCode(sharedSecret);
    }
    steamClient.logOn(logOnOptions);

    steamClient.on('loggedOn', () => {
      console.log('✅ Steam Bot logged in successfully');
      steamClient.setPersona(SteamUser.Steam.EPersonaState.Online);
      steamClient.gamesPlayed(730);
    });

    steamClient.on('error', (err) => {
      console.error('Steam Bot error:', err);
    });

    steamClient.on('webSession', (sessionID, cookies) => {
      manager.setCookies(cookies, (err) => {
```

```
          if (err) {
            console.error('Failed to set cookies for trade manager:', err);
            return;
          }
          console.log('Steam Bot: Web session established, cookies set');
          if (identitySecret) {
            steamCommunity.startConfirmationChecker(30000, identitySecret);
          }
        });
        steamCommunity.setCookies(cookies);
      });

      manager.on('newOffer', async (offer) => {
        console.log(`New trade offer #${offer.id} from
${offer.partner.getSteamID64()}`);
        if (offer.itemsToReceive.length > 0 && offer.itemsToGive.length ===
0) {
          try {
            await offer.accept();
            console.log('Offer accepted: received items from user.');
            io.emit('bot_event', { type: 'offer_accepted', offerId: offer.id
});
          } catch (err) {
            console.error('Failed to accept offer:', err);
          }
        } else {
          offer.decline((err) => {
            console.log('Unrelated offer declined.');
          });
        }
      });

      manager.on('sentOfferChanged', (offer, oldState) => {
        console.log(`Offer #${offer.id} state changed: ${offer.state}`);
        if (offer.state === TradeOfferManager.ETradeOfferState.Accepted) {
          console.log('Our sent offer was accepted by the other party.');
          io.emit('bot_event', { type: 'offer_sent_accepted', offerId:
offer.id });
        }
      });
    }

    function sendItemToBuyer(assetId, buyerSteamId) {
      const offer = manager.createOffer(buyerSteamId);
      offer.addMyItem({ appid: 730, contextid: 2, assetid: assetId });
      offer.setMessage("Trade from SteamTradingApp: delivering item");
      offer.send((err, status) => {
```

```
        if (err) {
          console.error('Error sending trade offer to buyer:', err);
        } else {
          console.log(`Trade offer sent to buyer. Status: ${status}`);
          if (status === 'pending') {
            steamCommunity.acceptConfirmationForObject(identitySecret,
offer.id, (err) => {
              if (err) console.error('Failed to confirm trade offer:', err);
              else   console.log('Trade   offer   confirmed   via   mobile
confirmation.');
            });
          }
        }
      });
    }

    module.exports = { init, sendItemToBuyer };

    import React from 'react';
    import { EyeIcon } from '@heroicons/react/24/outline';

    function ItemCard({ item, onBuy }) {
      let wearRatio = 0;
      if (item.exterior) {
        const ext = item.exterior;
        if (ext === "Factory New") wearRatio = 0.1;
        else if (ext === "Minimal Wear") wearRatio = 0.3;
        else if (ext === "Field-Tested") wearRatio = 0.5;
        else if (ext === "Well-Worn") wearRatio = 0.7;
        else if (ext === "Battle-Scarred") wearRatio = 0.9;
      }
      const views = Math.floor(Math.random() * 100) + 1;
      const priceChange = (Math.random() * 10 - 5).toFixed(1);

      return (
        <div className="bg-dark-card p-4 rounded shadow-md flex flex-col">
          <h3   className="font-semibold">{item.name}{item.exterior   ?   `
(${item.exterior})` : ''}</h3>
          <img  src={item.image}  alt={item.name}  className="my-2  max-h-40
object-contain" />
          <div className="flex items-center text-sm text-gray-400 mb-1">
            <EyeIcon className="h-4 w-4 inline-block mr-1" /> {views}
            <span className="ml-4">{priceChange}%</span>
          </div>
          {item.exterior && (
            <div className="w-full bg-gray-600 rounded h-2 mb-2">
              <div
```

```
                  className={`h-2 rounded ${wearRatio < 0.4 ? 'bg-green-500'
: wearRatio < 0.7 ? 'bg-yellow-500' : 'bg-red-500'}`}
                  style={{ width: `${wearRatio * 100}%` }}
                />
              </div>
            )}
            <div  className="text-lg  font-bold  mb-2">{item.price.toFixed(2)}
₴</div>
            {item.seller && (
              <div className="text-sm text-gray-300 mb-2">
                Продавець: {item.seller.displayName} –
                <span className="ml-1">{item.seller.tradeCount} угод</span>,
                <span className="ml-1">
                  <span className="text-red-500">офлайн</span>
                </span>
              </div>
            )}
            <button onClick={onBuy} className="mt-auto bg-blue-600 hover:bg-
blue-700 py-2 rounded">
              Купити
            </button>
          </div>
        );
      }

      export default ItemCard;
```