НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	**_____ Белла ГОЛУБ**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення для автоматизації торгівлі віртуальними предметами»

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Віктор БОРОВИК**
асистент (підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Юлія БОЯРІНОВА**
к.т.н., доцент (підпис)

Виконав _____ **Максим КУМПАН**
(підпис)

КИЇВ – 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Кумпану Максиму Вадимовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення для автоматизації торгівлі віртуальними предметами»
затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: опис процесів торгівлі віртуальними предметами на платформі Steam та сторонніх торгових майданчиках
Перелік питань, що підлягають дослідженню: дослідження предметної області та вимог, проєктування інформаційного забезпечення, проєктування та розробка програмного забезпечення системи, рекомендації щодо впровадження та експлуатації системи.
Перелік графічних документів (за необхідності).

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Віктор БОРОВИК**
(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Юлія БОЯРІНОВА**
(підпис)

Завдання взяв до виконання _____ **Максим КУМПАН**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- API – Application Programming Interface
- AES – Advanced Encryption Standard
- ER – Entity-Relationship
- JWT – JSON Web Token
- OAuth – Open Authorization
- SDK – Software Development Kit
- SPA – Single-Page Application
- SQL – Structured Query Language
- TLS – Transport Layer Security
- TOTP – Time-based One-Time Password
- UML – Unified Modeling Language
- ПЗ – програмне забезпечення
- СУБД – система управління базами даних

ВСТУП

Актуальність. Стрімкий розвиток індустрії багатокористувацьких онлайн-ігор призвів до формування окремого сегменту цифрової економіки – ринку віртуальних предметів. Сукупний обсяг операцій з віртуальними предметами на провідних торгових платформах обчислюється мільярдами доларів щорічно [1]. Користувачі здійснюють обмін, купівлю та продаж предметів як на офіційному маркетплейсі платформи Steam, так і на сторонніх торгових платформах, що пропонують ширші можливості пошуку та нижчі комісії.

Ручна торгівля віртуальними предметами стає неефективною через високу швидкість змін на ринку, велику кількість одночасно опрацьовуваних позицій та необхідність порівняння пропозицій з декількох платформ одночасно. Найвигідніші пропозиції зникають з ринку протягом секунд після появи, що робить ручну реакцію практично неможливою. Користувач, який торгує з кількома Steam-акаунтами одночасно, змушений виконувати рутинні операції підтвердження угод та керування інвентарем у ручному режимі для кожного акаунту окремо.

Існуючі рішення здебільшого зосереджені на окремих платформах або обмеженому наборі функцій і не пропонують комплексного інструменту для автоматизованої торгівлі віртуальними предметами, який поєднує керування інвентарем, моніторинг цін та автоматичне виконання операцій. Це обґрунтовує доцільність розробки нової програмної системи, що інтегрує перелічені можливості у єдиний інструмент.

Мета розробки. Мета розробки полягає в автоматизації процесів торгівлі віртуальними предметами та полегшенні щоденної роботи учасника ринку на основі створення програмного забезпечення для платформи Steam і сторонніх торгових майданчиків. Завдяки цьому можна буде керувати кількома обліковими записами з єдиного інтерфейсу, миттєво отримувати інформацію про ринкові ціни на різних платформах, не пропускати вигідні пропозиції, що

з'являються лише на секунди, та позбутися рутинних дій з ручного підтвердження угод і моніторингу інвентарю.

Основна ідея полягає в тому, щоб вирішити існуючі проблеми ручної торгівлі, коли користувач змушений тримати відкритими десятки вкладок, паралельно стежити за цінами на різних маркетплейсах і встигати вручну реагувати на зміни ринку. Розроблене програмне забезпечення дозволить централізовано зберігати дані про облікові записи й інвентарі, інтегрувати усі торгові операції в єдину систему та автоматично виявляти недооцінені пропозиції за параметрами зношеності і шаблону предмета. Це зробить торгівлю більш системною та продуктивною, скоротить час на рутинні операції і знизить ризик пропустити вигідну угоду через людський фактор.

Методи та технології. Методологічною основою роботи є принципи об'єктно-орієнтованого проєктування, клієнт-серверної архітектури, REST-взаємодії та сучасних підходів до побудови кросплатформних застосунків з єдиною кодовою базою. Як основну мову програмування обрано TypeScript, що забезпечує єдиний синтаксис клієнтської та серверної частин і дозволяє повторно використовувати спільні типи даних.

Клієнтська частина побудована на фреймворку SvelteKit 5.0 з бібліотекою UI-компонентів Shadcn-svelte та стилізацією через TailwindCSS. Для пакування у нативні застосунки для Windows, macOS, Linux та Android використано платформу Tauri 2.0, що дозволяє з єдиної кодової бази формувати десктопну, мобільну та вебверсії системи.

Серверну частину реалізовано на платформі Node.js з використанням фреймворку Express. Робота зі Steam Network виконується через спеціалізовані бібліотеки steam-user, steam-totp, steamcommunity, steam-tradeoffer-manager та globaloffensive, що забезпечують повноцінну взаємодію з ігровою платформою без офіційних обмежень Steam Web API. Для інтеграції зі сторонніми торговими платформами (CSFloat, Market.CSGO, Youpin) реалізовано окремі HTTP-клієнти, що нормалізують ринкові дані до єдиного внутрішнього формату.

Зберігання даних виконується у документоорієнтованій базі Google Cloud Firestore у складі екосистеми Firebase, що додатково надає сервіс автентифікації Firebase Authentication через Google OAuth та засіб виконання тригерних функцій Cloud Functions. Валідація даних на межі клієнт-сервер виконується за допомогою бібліотеки Zod, конфіденційні дані облікових записів Steam шифруються алгоритмом AES-256-GCM. Зовнішній доступ до серверної частини забезпечується через Cloudflare Tunnel без відкриття публічних портів на сервері.

Апробація програмного продукту. Результати розробки представлено на VII Всеукраїнській науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем», що відбулась 29 квітня 2026 року, у вигляді тез доповіді з назвою «Програмне забезпечення для автоматизації торгівлі віртуальними предметами».

Структура записки. Пояснювальна записка містить вступ, чотири основні розділи та висновки. До записки також входять перелік використаних джерел і додатки з графічним матеріалом. Загальний обсяг записки – 84 сторінки основного тексту, кількість використаних джерел – 21, кількість додатків – 6.

У першому розділі «Торгівля віртуальними предметами як об'єкт дослідження» розглянуто особливості ринку віртуальних предметів як предметної області, змодельовано основні процеси за допомогою UML-діаграм, проаналізовано існуючі рішення, а також сформульовано завдання, яке покликана вирішити розроблена система.

У другому розділі «Проєктування інформаційного забезпечення» побудовано логічну модель даних у формі ER-діаграми, обґрунтовано вибір системи управління базами даних, спроєктовано структуру колекцій інформаційної бази та її фізичну організацію.

У третьому розділі «Розробка програмного забезпечення» висвітлено етапи побудови прикладної частини системи: обґрунтовано вибір технологічного стеку, описано організаційну та компонентну архітектуру

застосунку, окремо розглянуто алгоритмізацію ключових модулів та фрагменти програмної реалізації.

Четвертий розділ «Рекомендації щодо впровадження та експлуатації системи» охоплює функціональне тестування реалізованих модулів, вимоги до апаратного й програмного забезпечення вузлів системи, перелік артефактів інсталяційного пакета та практичні поради щодо розгортання застосунку у робочому середовищі.

1 ТОРГІВЛЯ ВІРТУАЛЬНИМИ ПРЕДМЕТАМИ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

1.1 Аналіз торгівлі віртуальними предметами як предметної області

Ігрова платформа Steam – це онлайн-сервіс компанії Valve, який надає розробникам ігор готовий механізм видачі та зберігання внутрішньоігрових предметів, прив'язаних до облікового запису гравця (Steam Inventory Service) [2]. Якщо розробник дозволяє обмін предметами, останні набувають властивостей самостійних активів – мають ринкову вартість, можуть продаватися, обмінюватися та накопичуватися у вигляді колекцій. На цій основі сформувався окремий сегмент цифрової економіки – ринок віртуальних предметів, сукупний обсяг операцій якого на провідних торгових платформах обчислюється мільярдами доларів щорічно [1].

Купівля та продаж віртуальних предметів здійснюється двома способами: на офіційному маркетплейсі Steam Community Market, де торгівля відбувається у валюті гаманця Steam без виведення у реальні гроші, або на сторонніх торгових платформах, що приймають реальні платіжні засоби та пропонують ширші можливості пошуку, нижчі комісії та спеціалізовані фільтри за характеристиками предметів. Передача предмета між учасниками угоди завжди відбувається у самій системі Steam через механізм пропозиції обміну, а сторонні платформи виступають посередниками-брокерами: приймають оплату покупця на умовне зберігання, перевіряють факт передачі через Steam і лише після цього переводять кошти продавцю за вирахуванням комісії.

Віртуальні предмети мають індивідуальні характеристики, що суттєво впливають на ринкову вартість. Найважливіші з них – зношеність (float) – числове значення стану предмета від 0 (ідеальний) до 1 (максимально зношений), та шаблон (paint seed) – цілочисельний ідентифікатор унікального

малюнка забарвлення. Ці характеристики не є обов'язковими для всіх категорій, але саме їх поєднання формує найбільш цінні екземпляри, ціна яких може у десятки разів перевищувати середньоринкову.

Ринок характеризується високою динамікою. Ціни на однакові предмети відрізняються на різних платформах через відмінності у структурі комісій та доступних способах виведення коштів – це створює можливості для арбітражних операцій. Окрім того, постійно з'являються пропозиції з аномально низькою для певної комбінації характеристик ціною, що дозволяє отримувати прибуток оперативним виявленням і купівлею недооцінених лотів.

Ручне виконання торгових операцій є вкрай неефективним: найвигідніші пропозиції зникають з ринку протягом секунд, моніторинг цін на десятках платформ для тисяч предметів вручну неможливий, а робота з кількома Steam-акаунтами одночасно вимагає постійних рутинних дій з підтвердження угод і керування інвентарем. Це робить автоматизацію процесів актуальним завданням, що дозволяє суттєво підвищити ефективність торгової діяльності та зменшити вплив людського фактору на її результати.

1.2 Моделювання предметної області

Словесний опис предметної області не завжди дозволяє виявити неоднозначності у вимогах до системи та узгодити їх між замовником та розробником. Подолати ці труднощі допомагає формальне моделювання – подання предметної області у вигляді графічних діаграм, що зображають її учасників, їхні дії та порядок взаємодії [3].

Для опису торгівлі віртуальними предметами як предметної області використовується нотація UML (Unified Modeling Language) – загальноприйнятий галузевий стандарт графічного представлення моделей програмних систем [4]. Для опису предметної області використано три UML-діаграми: прецедентів, послідовності та активності – кожна з яких розкриває

окремий аспект взаємодії учасників торгівлі. Кожна з цих діаграм розглядається у подальших підрозділах.

1.2.1 Діаграма прецедентів. Діаграма прецедентів (use case diagram) призначена для опису функціональних можливостей системи або предметної області з точки зору її зовнішнього спостерігача [5]. Така діаграма фіксує не «як» виконуються дії, а «що» саме виконується – тобто перелік завдань, доступних кожному учаснику, без деталізації внутрішньої логіки їх реалізації. Завдяки своїй простоті діаграма прецедентів використовується на ранніх етапах аналізу предметної області, коли необхідно сформувати спільне розуміння обсягу системи між замовником та розробником.

Основними елементами діаграми є актори та прецеденти. Актором вважається будь-яка зовнішня сутність – особа, технічна система або підрозділ організації – яка взаємодіє з досліджуваною предметною областю та ініціює певні дії. Прецедент описує окрему функціональну можливість, доступну для виконання у відповідь на дії актора. Зв'язки між акторами та прецедентами зображаються у вигляді ліній асоціації, а між самими прецедентами можуть існувати спеціальні відношення включення (include) та розширення (extend), що дозволяють винести спільну поведінку в окремі прецеденти або описати додаткові сценарії, які виконуються лише за певних умов [5].

На рис. 1 представлено діаграму прецедентів, побудовану для предметної області торгівлі віртуальними предметами. Діаграма охоплює чотирьох учасників торгових операцій та основні дії, які кожен з них виконує у межах типового сценарію купівлі-продажу.

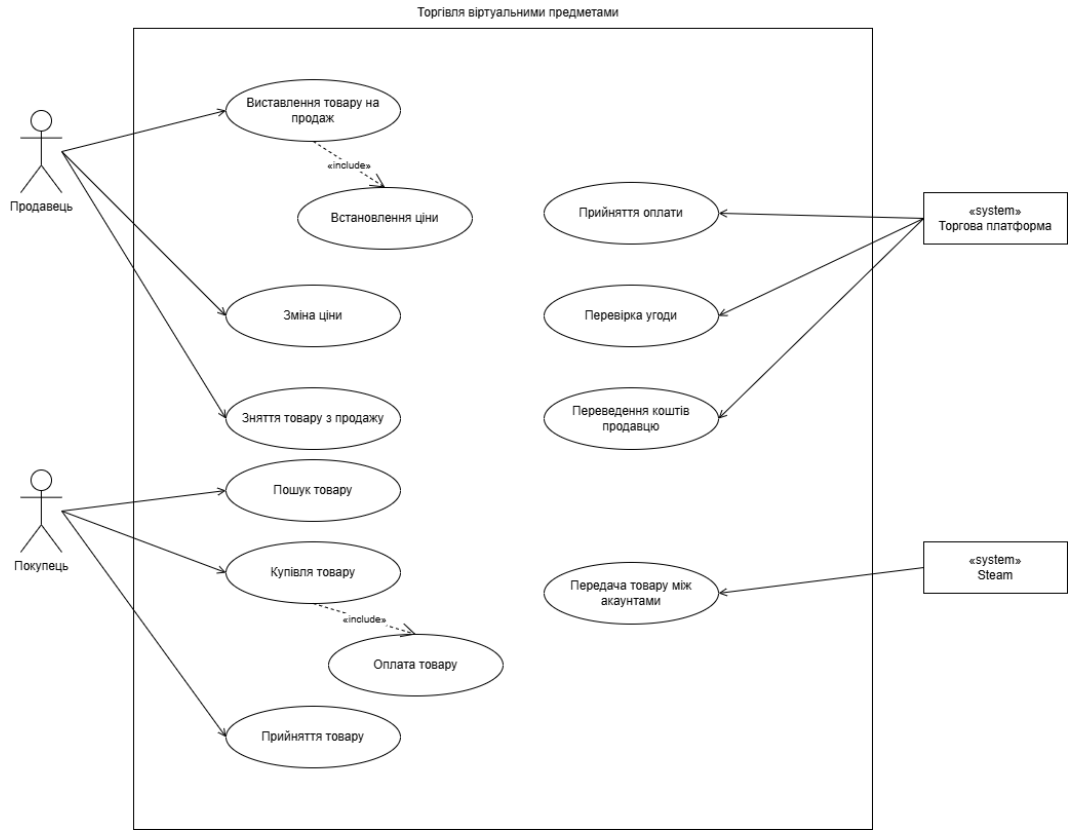

Рис. 1 Діаграма прецедентів предметної області торгівлі віртуальними предметами

Опис акторів

Було виокремлено чотирьох ключових акторів, які беруть участь у процесах предметної області.

- Продавець – фізична особа, яка володіє віртуальними предметами та має намір реалізувати їх через торгову платформу;
- Покупець – фізична особа, яка має намір придбати віртуальні предмети для подальшого використання, перепродажу або колекціонування;
- Торгова платформа – зовнішня система, що виконує функції майданчика для розміщення оголошень та брокера, який гарантує виконання умов угоди;
- Steam – зовнішня система-екосистема компанії Valve, у якій зберігаються віртуальні предмети та виконується безпосередня передача предметів між обліковими записами учасників угоди.

Опис прецедентів

Продавець:

- виставлення товару на продаж – розміщення предмета з власного інвентарю на торговій платформі (з обов'язковим включенням прецеденту «Встановити ціну»);
- зміна ціни – коригування ціни раніше виставленого предмета відповідно до зміни ринкової кон'юнктури;
- зняття товару з продажу – відкликання раніше розміщеного оголошення.

Покупець:

- пошук товару – перегляд переліку доступних предметів та фільтрація їх за заданими критеріями;
- купівля товару – ініціація угоди щодо обраного предмета (з обов'язковим включенням прецеденту «Оплатити товар»);
- прийняття товару – підтвердження отримання предмета в інвентар власного облікового запису Steam.

Торгова платформа:

- прийняття оплати – резервування коштів покупця на умовному зберіганні до моменту завершення угоди;
- перевірка угоди – верифікація факту успішної передачі предмета між обліковими записами учасників;
- переведення коштів продавцю – перерахування грошових коштів за вирахуванням комісії після підтвердження виконання угоди.

Steam:

- передача товару між акаунтами – виконання операції переміщення предмета з інвентарю продавця до інвентарю покупця через механізм пропозиції обміну.

Побудована діаграма прецедентів показує, які саме дії охоплює предметна область торгівлі віртуальними предметами та як обов'язки розподілені між її учасниками. Це створює основу для подальшого моделювання – діаграми послідовності, що відображає часовий аспект

взаємодій, та діаграми активності, що описує внутрішню логіку процесів продажу і купівлі.

1.2.2 Діаграма послідовності. На відміну від діаграми прецедентів, що відповідає на питання «що може бути виконано», діаграма послідовності (sequence diagram) відповідає на питання «у якому порядку це виконується» [6]. Цей тип діаграм належить до групи поведінкових діаграм UML і призначений для опису конкретного сценарію виконання – впорядкованого у часі обміну повідомленнями між учасниками процесу. Діаграма послідовності використовується тоді, коли необхідно деталізувати один з прецедентів, виявити ролі окремих компонентів у його реалізації та зафіксувати протокол їх взаємодії.

Структурно діаграма послідовності складається з вертикальних ліній життєвого циклу, що відповідають учасникам сценарію, та горизонтальних стрілок повідомлень, що з'єднують ці лінії. Часова шкала діаграми спрямована згори донизу: повідомлення, розташовані вище, передаються раніше; повідомлення, розташовані нижче, – пізніше. На лінії життєвого циклу можуть відображатися смуги активації – прямокутні фрагменти, що позначають періоди, протягом яких учасник виконує певну дію у відповідь на отримане повідомлення. Між учасниками можуть існувати як синхронні повідомлення (відправник очікує відповіді), так і асинхронні (відправник продовжує виконання незалежно від отримувача).

У Додатку А представлено діаграму послідовності, яка ілюструє типовий сценарій взаємодії учасників предметної області у процесі продажу та купівлі віртуального товару. Відображено повний цикл угоди – від моменту виставлення товару продавцем до завершення передачі коштів за вирахуванням комісії платформи.

Опис акторів та об'єктів діаграми

- Продавець – ініціює виставлення предмета на продаж та формує пропозицію обміну в Steam після надходження замовлення;

- Торгова платформа – приймає лот, координує угоду, тримає оплату на умовному зберіганні та виконує розрахунок з продавцем після завершення передачі;
- Покупець – здійснює пошук та купівлю товару, оплачує його та приймає пропозицію обміну від продавця;
- Steam – забезпечує безпосередню передачу віртуального предмета між обліковими записами учасників угоди.

Опис сценарію

Розміщення товару на платформі:

- продавець виставляє товар на продаж на торговій платформі;
- платформа підтверджує прийняття лоту.

Пошук та купівля товару:

- покупець здійснює пошук товару у каталозі платформи;
- платформа повертає інформацію про обраний товар;
- покупець ініціює купівлю та здійснює оплату;
- платформа резервує кошти на умовному зберіганні та сповіщає продавця про факт продажу.

Передача товару через Steam:

- продавець створює пропозицію обміну на користь покупця в системі Steam;
- Steam сповіщає покупця про надходження пропозиції обміну;
- покупець приймає пропозицію обміну та підтверджує її засобом двофакторної автентифікації;
- Steam повідомляє торгову платформу про факт успішної передачі предмета.

Завершення угоди:

- платформа переводить кошти продавцю за вирахуванням комісії;
- платформа підтверджує покупцю успішне завершення угоди.

Наведений сценарій деталізує часову логіку процесу торгової операції та робить очевидною роль кожного з учасників у її виконанні. Зокрема, чітко

простежується розподіл відповідальності між торговою платформою (фінансова частина угоди) та екосистемою Steam (фактична передача предмета), що було ключовою особливістю предметної області, виявленою на попередньому етапі аналізу.

1.2.3 Діаграма активності. Діаграма активності (activity diagram) у нотації UML призначена для опису потоку керування у межах процесу [5]. На відміну від діаграми послідовності, яка фокусується на обміні повідомленнями між учасниками, діаграма активності робить акцент на самих діях та умовах переходу між ними.

Основними елементами діаграми є: початковий вузол (зображається суцільним чорним колом), що позначає старт процесу; кінцевий вузол (чорне коло, обведене кільцем), що позначає його завершення; вузли дій у вигляді прямокутників із заокругленими кутами; стрілки потоку керування, що з'єднують ці вузли; та вузли прийняття рішення у вигляді ромбів, які описують розгалуження процесу залежно від виконання певної умови.

Розроблену діаграму активності, що моделює потоки керування під час продажу та купівлі віртуального товару, винесено у Додаток Б. На діаграмі виокремлено два паралельні потоки керування – процес продажу з точки зору продавця та процес купівлі з точки зору покупця, – що відповідає природному розділенню ролей у предметній області.

Опис діаграми

Процес продажу віртуального товару (потік продавця):

- продавець обирає товар з власного інвентарю та встановлює ціну;
- товар виставляється на торгову платформу;
- якщо лот не пройшов валідацію – продавець виправляє дані лоту та повторює спробу;
- після прийняття лоту продавець очікує замовлення від покупця;
- при отриманні сповіщення про продаж продавець створює пропозицію обміну в Steam;

- продавець підтверджує передачу через двофакторну автентифікацію (у разі неуспішного підтвердження – звертається до служби підтримки);
- після успішної передачі продавець очікує зарахування коштів та отримує їх за вирахуванням комісії.

Процес купівлі віртуального товару (потік покупця):

- покупець здійснює пошук товару на торговій платформі (якщо товар не підходить – продовжує пошук далі);
- покупець оформлює замовлення на обраний предмет;
- покупець оплачує товар (у разі невдалої оплати – повторює спробу);
- покупець очікує пропозицію обміну від продавця;
- при надходженні пропозиції обміну покупець приймає її та підтверджує засобом двофакторної автентифікації;
- покупець отримує товар до власного інвентарю Steam.

Подане розгалуження дій за двома виконавчими ролями та явне виокремлення альтернативних шляхів виконання (помилки валідації, повторні спроби оплати, неуспішне підтвердження) робить діаграму активності зручним джерелом для подальшого формулювання сценаріїв нормального та виняткового перебігу подій у програмній системі. Особливо цінним є те, що діаграма явно фіксує ті ситуації, у яких від користувача очікується ручне втручання – саме ці ділянки процесу підлягатимуть автоматизації у системі, що розробляється.

1.3 Огляд існуючих аналогічних систем

В умовах активного розвитку ринку віртуальних предметів виникла потреба у програмних інструментах, які дозволяють учасникам ринку автоматизувати окремі етапи торгової діяльності, відстежувати ціни на численних торгових платформах, керувати інвентарем кількох облікових записів та виявляти прибуткові операції. Серед найбільш відомих програмних рішень у цій сфері виділяються Skinledger, TradeOn Ultra та SteamLedger. Далі

по черзі описано кожне з рішень із виокремленням сильних і слабких сторін з погляду задач автоматизованої торгівлі.

Skinledger

Skinledger – одне з найбільш функціональних рішень для відстеження вартості віртуальних предметів у складі інвентарю користувача та керування Steam-акаунтами. Платформа складається з вебзастосунку та десктопного клієнта, що працюють синхронно та дозволяють користувачу мати єдиний доступ до власних активів незалежно від пристрою. Skinledger підтримує одночасний вхід у декілька облікових записів Steam через єдиний інтерфейс, надає функції оцінки сукупної вартості інвентарю, побудови графіка зміни цієї вартості з часом, керування пропозиціями обміну та відстеження цін з кількох торгових платформ [7].

На рис. 2 представлено інтерфейс системи Skinledger.

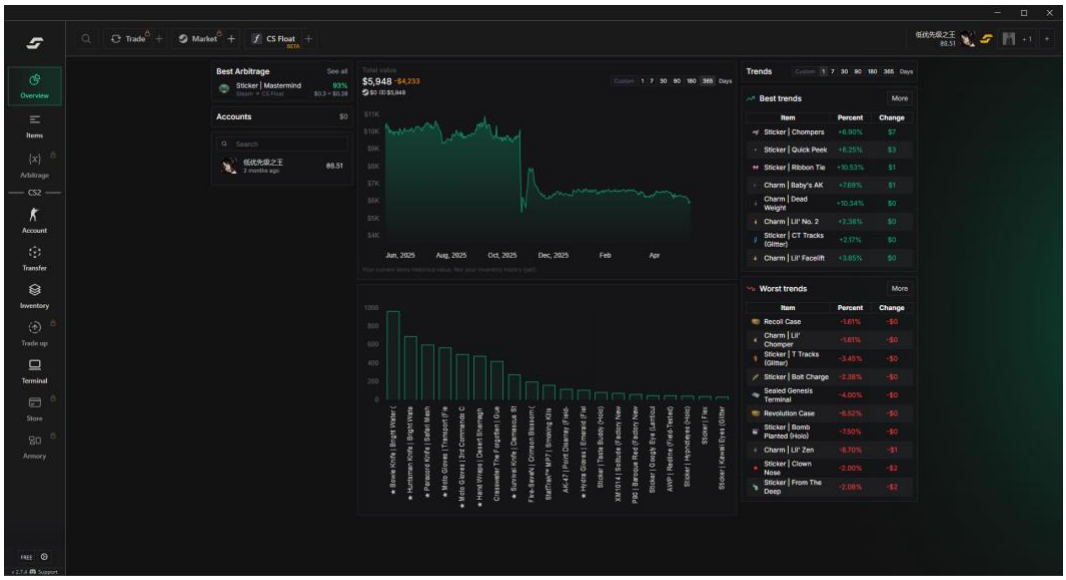

Рис. 2 Інтерфейс системи Skinledger

Переваги:

- наявність як вебзастосунку, так і десктопного клієнта з синхронізацією даних;
- одночасний вхід у декілька облікових записів Steam через єдиний інтерфейс;

- інтеграція з кількома джерелами цінових даних з різних торгових платформ;
- візуалізація динаміки вартості інвентарю у вигляді графіка.

Недоліки:

- управління кількома обліковими записами здійснюється відокремлено – кожен акаунт користувач веде як окремий, без можливості координованих операцій (зокрема перенесення предметів між обліковими записами користувача);
- відсутність історії власних угод з фіксацією цін купівлі та продажу – користувач не має змоги переглянути, скільки коштував той чи інший предмет на момент придбання;
- статистика обмежується графіком сукупної вартості інвентарю, відсутні детальні звіти про результативність окремих торгових операцій;
- відсутність функції автоматичного виявлення недооцінених лотів за параметрами зношеності та шаблону.

TradeOn Ultra

TradeOn Ultra – автоматизована торгова вебпанель для роботи з віртуальними предметами на платформі Steam та сторонніх торгових майданчиках. Система побудована як єдиний вебінтерфейс, у якому користувач може налаштувати арбітражні схеми – правила, що визначають купівлю предмета на одній торговій платформі за нижчою ціною та автоматичний перепродаж на іншій платформі за вищою. Після одноразового налаштування таких схем торгові операції виконуються автоматично без участі користувача за заданими параметрами цінової різниці, цільових платформ та категорій предметів [8]. До складу платформи входить вбудований менеджер облікових записів з безкоштовним аналогом мобільного застосунку двофакторної автентифікації Steam (Steam Desktop Authenticator), що дозволяє автоматично підтверджувати пропозиції обміну.

На рис. 3 показано інтерфейс системи TradeOn Ultra.

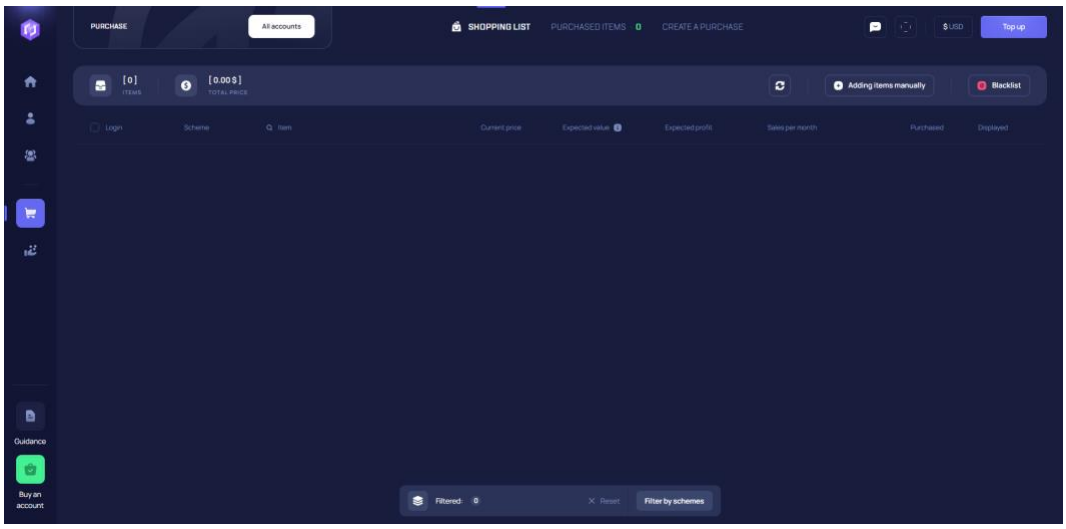

Рис. 3 Інтерфейс системи TradeOn Ultra

Переваги:

- повна автоматизація купівлі та продажу за налаштованими арбітражними схемами без участі користувача;
- вбудований безкоштовний менеджер облікових записів з підтримкою двофакторної автентифікації;
- робота у вікні браузера без необхідності встановлення додаткового програмного забезпечення;
- детальна статистика виконаних операцій з розрахунком прибутковості.

Недоліки:

- система зосереджена виключно на арбітражних операціях між торговими платформами – відсутні інші типи торгових алгоритмів, зокрема автоматичне виявлення недооцінених лотів за параметрами зношеності та шаблону;
- відсутність десктопного та мобільного клієнтів – система доступна виключно через браузер;
- відсутність функцій координованого керування кількома обліковими записами Steam, зокрема перенесення предметів між акаунтами користувача.

SteamLedger

SteamLedger – безкоштовний інструмент для відстеження портфеля віртуальних предметів та пошуку арбітражних можливостей. Система сканує понад 15 000 найменувань предметів на 11 торгових платформах щогодини, виявляючи цінові різниці та формуючи перелік пропозицій з прибутком після врахування комісій торгових майданчиків. Окрім сканера арбітражу, SteamLedger надає функції оцінки інвентарю, відстеження повернення інвестицій, побудови історичних графіків цін та аналізу ефективності власних торгових операцій [9].

Інтерфейс системи SteamLedger наведено на рис. 4.

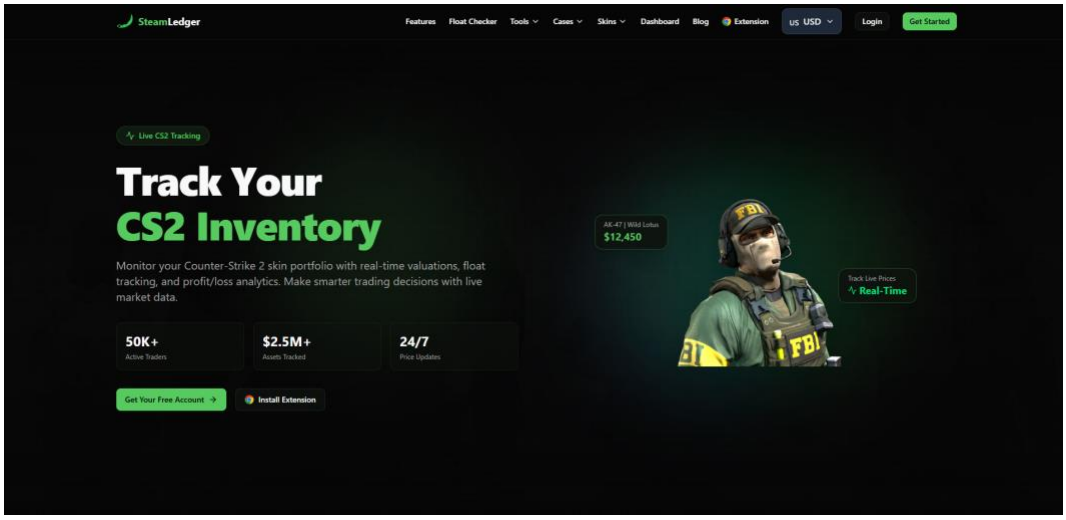

Рис. 4 Інтерфейс системи SteamLedger

Переваги:

- безкоштовний доступ до основного функціоналу;
- моніторинг 11 торгових платформ одночасно з обчисленням прибутку з урахуванням комісій;
- детальна аналітика портфеля з історичними графіками цін за тривалий період;
- експорт даних у формат CSV для подальшої обробки або податкової звітності.

Недоліки:

- система виявляє арбітражні можливості, але не виконує операції автоматично – користувач має здійснювати купівлю та продаж вручну;

- оновлення даних відбувається з періодичністю в одну годину, що недостатньо для високочастотних торгових операцій;
- відсутність функції автоматичного виявлення недооцінених лотів за параметрами зношеності та шаблону;
- відсутність функцій управління кількома обліковими записами Steam та автоматизації операцій обміну;
- відсутність автоматичного завантаження ціни придбання предметів – користувач має вручну вносити суму купівлі для кожного предмета, що ускладнює облік портфеля та обчислення фактичного прибутку.

Незважаючи на функціональні переваги розглянутих систем, жодна з них не забезпечує повного циклу автоматизованої торгівлі, який поєднує одночасну роботу з декількома обліковими записами Steam, моніторинг цін з кількох торгових платформ у режимі реального часу, спеціалізовані алгоритми пошуку лотів за характеристиками предметів та автоматичне виконання купівлі-продажу без участі користувача. Skinledger орієнтований переважно на облік вартості інвентарю в межах одного облікового запису, TradeOn Ultra – на виконання базових схем купівлі та продажу через вебпанель з обмеженим набором платформ, SteamLedger – на пошук арбітражних можливостей у режимі ручного виконання. Це обґрунтовує доцільність розробки нового програмного забезпечення, яке інтегрує ці окремі функції у єдиний інструмент автоматизованої торгівлі віртуальними предметами.

1.4 Постановка завдання

Метою даного дипломного проєкту є розробка програмного забезпечення для автоматизації торгівлі віртуальними предметами на платформі Steam та сторонніх торгових платформах. Система повинна забезпечувати повний цикл роботи з торговими даними: від керування обліковими записами та інвентарем користувача до отримання інформації про ринкові ціни та автоматичного виявлення вигідних пропозицій на ринку. Центральним функціональним

модулем системи має стати реалізація алгоритму автоматичного виявлення недооцінених пропозицій на ринку за параметрами зношеності та шаблону віртуальних предметів.

Вхідні дані системи

Система повинна обробляти такі вхідні дані:

- дані облікового запису користувача: ідентифікатор, електронна адреса;
- дані Steam-акаунтів користувача: ідентифікатор облікового запису, логін, файл двофакторної автентифікації (maFile), обмінне посилання користувача, тип акаунту (торговий або сховище);
- дані інвентарю: перелік віртуальних предметів з характеристиками (назва, категорія, зношеність, шаблон, наклейки, статус обмеження торгових операцій);
- дані ринкових пропозицій: лоти з торгових платформ з ціною, характеристиками предмета, ідентифікатором продавця та часом розміщення;
- правила купівлі недооцінених пропозицій: назва цільового предмета, діапазон значень зношеності та шаблону, максимально допустима ціна, перелік цільових торгових платформ;
- історія угод: записи про здійснені купівлі та продажі з датами, цінами, ідентифікаторами платформ та результатами.

Вихідні дані системи

Система повинна генерувати такі результати:

- візуальне представлення інвентарю користувача з агрегованими даними з усіх облікових записів та цінами з кількох торгових платформ;
- перелік виявлених недооцінених лотів з розрахунком потенційного прибутку;
- статус виконання активних правил купівлі недооцінених пропозицій;
- історія транзакцій з розрахунком чистого прибутку з урахуванням комісій торгових платформ;

- аналітичні графіки загальної прибутковості торгових операцій за вибраний період у вигляді стовпчиків по днях або місяцях.

Функціональні вимоги до системи

Реєстрація та автентифікація користувачів. Система повинна підтримувати реєстрацію нових користувачів та автентифікацію існуючих через хмарний сервіс ідентифікації. Доступ до даних повинен бути обмежений правами власника облікового запису, а конфіденційні дані (файли двофакторної автентифікації Steam) повинні зберігатися у зашифрованому вигляді.

Керування обліковими записами Steam. Користувач повинен мати можливість додавати та керувати кількома обліковими записами Steam через єдиний інтерфейс. Підтримується імпорт облікових записів через файл двофакторної автентифікації (maFile) з автоматичною генерацією одноразових кодів для підтвердження торгових операцій.

Синхронізація інвентарю. Система повинна автоматично отримувати інформацію про віртуальні предмети з облікових записів Steam, зберігати її у локальному кеші та забезпечувати її актуалізацію за запитом користувача або за розкладом.

Виставлення предметів на продаж. Користувач повинен мати можливість виставляти предмети з власного інвентарю на продаж на підтримуваних торгових платформах одночасно з єдиного інтерфейсу із заданою ціною.

Передача предметів між обліковими записами. Система повинна забезпечувати створення пропозицій обміну між власними обліковими записами користувача та за обмінним посиланням з автоматичним підтвердженням операцій засобом двофакторної автентифікації.

Налаштування правил купівлі недооцінених пропозицій. Користувач повинен мати можливість створювати правила автоматичної купівлі для конкретних віртуальних предметів. Кожне правило містить назву цільового предмета, діапазон допустимих значень зношеності та шаблону, максимально допустиму ціну та перелік цільових торгових платформ. Після збереження правила система починає відстежувати поточні пропозиції на ринку, а у разі

появи лоту, що відповідає всім заданим умовам, автоматично здійснює його купівлю.

Аналітика та звітність. Система повинна формувати звіти про виконані торгові операції, чистий прибуток з урахуванням комісій торгових платформ та графік загальної прибутковості за обраний період.

Нефункціональні вимоги до системи

Безпека. Конфіденційні дані облікових записів Steam (файли двофакторної автентифікації) повинні зберігатися виключно у зашифрованому вигляді. Доступ до даних користувача має бути обмежений правилами захисту даних на рівні хмарного сховища, що унеможливлює доступ до них з боку інших користувачів системи.

Продуктивність. Модуль автоматичного виявлення недооцінених пропозицій повинен реагувати на появу нового лоту на ринку з мінімально можливою затримкою (не більше декількох секунд від моменту появи лоту до спроби його купівлі).

Зручність використання. Інтерфейс користувача повинен бути зрозумілим для учасників ринку з мінімальним технічним досвідом. Основні функції повинні бути доступними з головного меню.

Кросплатформність. Програмне забезпечення повинно бути доступним у трьох форматах поставки: десктопний застосунок для операційних систем Windows, macOS та Linux, мобільний застосунок та вебдоступ через браузер. Всі три формати повинні мати єдину кодову базу клієнтської частини.

Масштабованість. Архітектура системи повинна підтримувати горизонтальне масштабування у частині обробників, що взаємодіють зі Steam та сторонніми торговими платформами, для забезпечення можливості одночасної роботи з великою кількістю облікових записів та виконання значного обсягу торгових операцій.

Надійність. Система повинна стійко обробляти тимчасові збої програмних інтерфейсів торгових платформ та Steam із автоматичним

повторенням невдалих запитів. Відмова окремого обробника не повинна впливати на роботу інших компонентів системи.

Запропонована програмна система орієнтована на повну автоматизацію типових операцій учасника ринку віртуальних предметів. Завдяки впровадженню алгоритму автоматичного виявлення недооцінених пропозицій, підтримці одночасної роботи з кількома обліковими записами Steam та інтеграції з кількома торговими платформами, програмне забезпечення стане ефективним інструментом для підвищення продуктивності торгової діяльності та зменшення впливу людського фактору на її результати.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних описує предметну область з точки зору того, які саме інформаційні об'єкти існують у системі, які характеристики вони мають та як пов'язані між собою. Її особливість полягає у тому, що вона не прив'язана до конкретної технології зберігання – одна й та сама логічна модель може бути реалізована як у реляційній базі даних, так і у документоорієнтованому сховищі. Графічним представленням логічної моделі є ER-діаграма (від англ. Entity-Relationship – «сутність-зв'язок»), на якій сутності зображають у вигляді прямокутників з переліком атрибутів, а зв'язки між ними – у вигляді ліній з позначенням кратності [10].

Для розроблюваної системи автоматизації торгівлі віртуальними предметами виокремлено п'ять ключових сутностей, що відповідають основним інформаційним об'єктам предметної області: користувачі системи, прив'язані до них Steam-акаунти, віртуальні предмети у складі інвентарів цих акаунтів, журнал подій інвентаря (продажі, купівлі, отримання предмета з гри) та правила автоматичної купівлі недооцінених пропозицій. Зв'язки між сутностями реалізовано через зовнішні ключі, що забезпечує цілісність даних на логічному рівні.

ER-діаграма, що ілюструє розроблену логічну модель, наведена у Додатку В.

Опис сутностей та атрибутів

Нижче наведено опис кожної з виокремлених сутностей. Поля, виділені нижнім підкресленням, позначають структуровані атрибути – вкладені об'єкти або масиви об'єктів, що описують ієрархічно пов'язані з основною сутністю дані [11].

USERS (Користувач) – коренева сутність:

- id – унікальний ідентифікатор користувача (відповідає ідентифікатору сервісу автентифікації);
- gift_codes – масив непогашених подарункових кодів Steam, доступних для розподілу між акаунтами користувача;
- updated_at – дата останнього оновлення документа.

ACCOUNTS (Steam-акаунт) – дочірня сутність:

- id – унікальний ідентифікатор акаунта;
- user_id (FK) – посилання на власника акаунта;
- type – тип акаунта (основний, сховище, бот);
- name – відображувана назва акаунта;
- trade_url – обмінне посилання користувача для прийому пропозицій обміну;
- credentials – вкладений об'єкт з обліковими даними акаунта (логін, пароль, секрети двофакторної автентифікації);
- platforms – вкладений об'єкт з API-ключами та балансами на чотирьох підтримуваних торгових платформах;
- avatar, level, market_ban_lifted_at – допоміжні поля профілю Steam та статус обмеження торгових операцій.

ITEMS (Віртуальний предмет) – дочірня сутність:

- asset_id – природний ключ предмета з системи Steam, який гарантує унікальність на рівні бази;
- account_id (FK) – посилання на акаунт, в інвентарі якого знаходиться предмет;
- market_hash_name – ринкова назва предмета;
- icon – URL зображення предмета;
- type – категорія предмета (skin, case, sticker, charm тощо);
- float_value – числове значення зношеності (0–1, для скінів);
- paint_seed – цілочисельний шаблон забарвлення;
- tradable_at – момент часу, після якого предмет стає доступним для обміну;

- listings – вкладений масив активних виставлень предмета на торгових платформах;
- acquisition – вкладений об'єкт, що описує спосіб отримання предмета (ігрове отримання, купівля, обмін).

EVENTS (Подія) – дочірня сутність:

- id – детермінований ідентифікатор події, який утворюється з її семантичних параметрів (тип, акаунт, предмет, час);
- account_id (FK) – посилання на акаунт, у межах якого сталася подія;
- asset_id (FK) – посилання на предмет, з яким пов'язана подія;
- strategy_id (FK, опц.) – посилання на правило купівлі, що ініціювало подію (для автоматичних купівель);
- kind – тип події (продаж, купівля, ігрове отримання);
- occurred_at – момент часу настання події;
- price, cost_basis_usd, platform – фінансові показники операції та торгова платформа;
- item_snapshot – вкладений об'єкт зі знімком характеристик предмета на момент події.

STRATEGIES (Правило купівлі) – дочірня сутність:

- id – унікальний ідентифікатор правила;
- user_id (FK) – посилання на користувача-власника правила;
- name – назва правила, задана користувачем;
- target_market_hash_name – ринкова назва цільового віртуального предмета;
- conditions – вкладений масив наборів умов спрацювання (кожен набір включає діапазон зношеності, перелік бажаних значень шаблону, максимальну ціну та цільові торгові платформи);
- is_active – ознака активності правила;
- updated_at – дата останньої зміни.

Опис зв'язків

- Один користувач може володіти багатьма Steam-акаунтами (зв'язок 1:N), реалізовано через зовнішній ключ user_id у сутності ACCOUNTS;
- Один користувач може налаштувати багато правил купівлі (зв'язок 1:N), реалізовано через зовнішній ключ user_id у сутності STRATEGIES;
- Один Steam-акаунт може містити багато віртуальних предметів (зв'язок 1:N), реалізовано через зовнішній ключ account_id у сутності ITEMS;
- Один Steam-акаунт може породжувати багато подій (зв'язок 1:N), реалізовано через зовнішній ключ account_id у сутності EVENTS;
- Один віртуальний предмет може породжувати багато подій (зв'язок 1:N), реалізовано через зовнішній ключ asset_id у сутності EVENTS. Цей зв'язок є опціональним з боку EVENTS, оскільки після видалення предмета з інвентарю історія подій повинна залишатися доступною;
- Одне правило купівлі може опціонально ініціювати багато подій купівлі (зв'язок 0..1:N), реалізовано через зовнішній ключ strategy_id у сутності EVENTS. Опціональність дозволяє розрізняти автоматичні купівлі, виконані за правилом, від ручних купівель користувача.

Особливості логічної моделі

Розроблена логічна модель не належить до класичних реляційних структур. Це обумовлено специфікою предметної області:

- значна частина даних має складну ієрархічну структуру – облікові дані Steam-акаунта, набори API-ключів торгових платформ, активні виставлення предметів, способи отримання предметів – і не може бути зведена до плоских атомарних атрибутів без втрати семантичної цілісності;
- вкладена структура атрибутів безпосередньо відповідає природній організації даних у системах постачальників: інформація Steam про предмет надходить як цілісний документ з ієрархічними полями;
- у системі переважають операції читання усіх даних користувача та його інвентарю за одним запитом, а складні аналітичні запити з багаторівневими з'єднаннями таблиць не передбачені.

За наведеними ознаками подана модель не є реляційною. Обґрунтування вибору конкретної системи зберігання детально розглянуто у наступному підрозділі.

2.2 Вибір системи управління базами даних

Перш ніж розглядати конкретні системи зберігання, варто пояснити, чому обрано саме документоорієнтовану базу даних, а не класичну реляційну. Для цього корисно зрозуміти, як ці типи баз даних влаштовані.

Реляційна база даних – це сховище, у якому дані зберігаються у вигляді таблиць з фіксованою структурою. Кожна таблиця має чітко визначений перелік стовпців, а зв'язок між сутностями реалізується через відповідність значень у різних таблицях. Прикладом такої системи є PostgreSQL або MySQL. Документо-орієнтована база даних замість таблиць використовує колекції документів. Кожен документ – це структурований запис, схожий на JSON-об'єкт, який може містити будь-які поля, у тому числі вкладені об'єкти та масиви. Прикладами таких систем є Google Cloud Firestore та MongoDB.

Для розроблюваної системи обрано саме документоорієнтовану модель з трьох основних причин. По-перше, дані про користувача мають природну ієрархічну структуру – акаунт містить вкладені облікові дані, ключі від платформ, баланси на цих платформах. У реляційній моделі цю структуру довелось би розбивати на кілька окремих таблиць та з'єднувати їх при кожному запиті. У документоорієнтованому сховищі усі ці дані зберігаються разом у одному документі і отримуються однією операцією читання. По-друге, документоорієнтовані бази дозволяють легко змінювати структуру даних під час розробки – не потрібно виконувати міграції схеми, як це робиться у реляційних системах. По-третє, обрана платформа Firebase надає не лише саму базу, а ще й готовий сервіс автентифікації та засіб для виконання тригерних функцій у відповідь на події бази, що зменшує загальну кількість зовнішніх залежностей системи.

Розглянуто три варіанти систем зберігання, що відповідають документоорієнтованій або документо-сумісній моделі: Google Cloud Firestore, MongoDB Atlas та PostgreSQL з підтримкою JSONB. Порівняльна характеристика наведена у табл. 2.1.

Таблиця 2.1

Порівняльна характеристика розглянутих систем зберігання

Параметр	**Firestore**	**MongoDB Atlas**	**PostgreSQL (JSONB)**
Тип моделі	Документо-орієнтована	Документо-орієнтована	Реляційна з JSONB-полями
Інтеграція з автентифікацією	Firebase Authentication	Окремі сервіси	Окремі сервіси
Декларативні правила доступу	Firestore Security Rules	Через middleware	Через row-level security
Безсерверні функції-тригери	Cloud Functions	Atlas Triggers	Не передбачено
Автоматичне масштабування	Так	Так	Обмежене
Підтримка мобільних SDK	Повна	Через сторонні SDK	Не передбачено

За результатами аналізу обрано Google Cloud Firestore як основну СУБД проєкту. Цей вибір обґрунтовано наступними міркуваннями.

Інтеграція з екосистемою Firebase

Firestore є частиною платформи Firebase, що включає інші взаємодіючі сервіси: Firebase Authentication для автентифікації користувачів та Cloud Functions для реалізації серверної бізнес-логіки та тригерів подій бази. Використання єдиного провайдера всіх перелічених сервісів спрощує конфігурацію проєкту, забезпечує наскрізну автентифікацію та зменшує загальну кількість зовнішніх залежностей системи [12].

Декларативні правила безпеки доступу

Firestore Security Rules дозволяють описати правила доступу до документів безпосередньо на стороні бази даних, без необхідності маршрутизації запитів через проміжний сервер. Це усуває окремий клас вразливостей, пов'язаних з помилками в логіці контролю доступу на стороні клієнта.

Безсерверне виконання тригерів

Платформа Firebase Cloud Functions дозволяє виконувати фрагменти серверної логіки у відповідь на події бази (створення, оновлення, видалення документа). Це функціональний аналог тригерів класичних СУБД, який у поєднанні з декларативними правилами доступу забезпечує цілісність даних на рівні самого сховища.

Прозоре масштабування

Firestore автоматично адаптується до зростання навантаження – зі збільшенням кількості користувачів та обсягу даних розробнику не потрібно вручну налаштовувати розподіл навантаження між серверами. Це особливо цінно на ранніх етапах експлуатації системи, коли точна модель навантаження ще не сформована.

У структурі розроблюваного проєкту задіяно три сервіси Firebase: Firestore (зберігання даних), Firebase Authentication (автентифікація користувачів) та Cloud Functions (виконання тригерів). Усі ці сервіси належать одному проєкту Firebase, що забезпечує наскрізну ідентифікацію користувачів та узгоджене застосування правил доступу.

2.3 Розробка структури інформаційної бази

На цьому етапі визначається, як саме логічна модель буде представлена у фізичному сховищі. Інформаційна база розроблюваної системи складається з чотирьох основних компонентів: колекцій документів (вони відіграють роль таблиць у класичних базах даних), композитних індексів для прискорення типових запитів, тригерних функцій для підтримки цілісності даних та правил безпеки доступу. Реалізацію перелічених компонентів детально розкрито у відповідних пунктах нижче.

2.3.1 Створення проєкту Firebase та ініціалізація бази даних. Створення інформаційної бази в Firestore починається з реєстрації проєкту в Firebase Console. На рис. 5 наведено мінімальну конфігурацію проєкту у файлі .firebaserc, що пов'язує локальний код з певним проєктом Firebase.

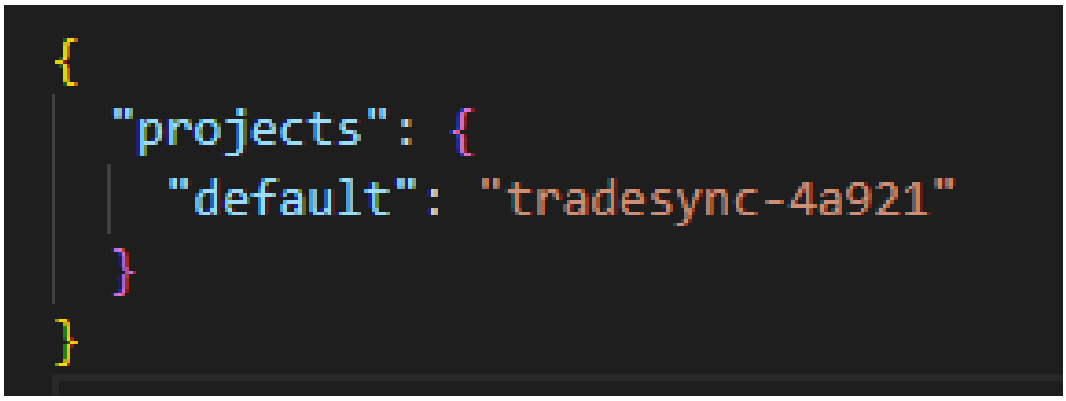

Рис. 5 Конфігурація проєкту Firebase у файлі .firebaserc

Після створення проєкту в його межах активовано три сервіси: Firestore (база даних), Firebase Authentication (автентифікація) та Cloud Functions (безсерверні функції-тригери). Активація сервісів виконується через Firebase Console одноразово і не потребує жодних команд у коді.

2.3.2 Створення колекцій. У Firestore колекція не оголошується явно – вона починає існувати у момент створення першого документа у ній. Тому структура документів задається не на рівні бази, а у коді клієнтського або серверного застосунку. Для цього використано бібліотеку Zod – інструмент для опису та валідації даних у середовищі TypeScript [13]. Zod-схема визначає перелік полів документа, типи їх значень та правила валідації перед записом.

На прикладі двох ключових колекцій – акаунтів та віртуальних предметів – розглянемо процес визначення структури. Фрагмент Zod-схеми для документа колекції accounts наведено на рис. 6, а для колекції items – на рис. 7.

Рис. 6 Zod-схема документа колекції accounts

```
export const itemSchema = z.object({
    assetId: z.string(),
    accountId: z.string(),

    marketHashName: z.string(),
    icon: z.url(),
    type: itemTypeSchema,
    game: gameSchema,

    tradable: z.number().nullable(),

    inspectionLink: z.string().nullable(),

    floatValue: z.number().nullable(),
    paintSeed: z.number().nullable(),

    listings: z.array(itemListingSchema).default([]),

    acquisition: acquisitionSchema,
});
```

Рис. 7 Zod-схема документа колекції items

Усі підколекції користувача (accounts, items, events, strategies) розташовані під документом users/{userId}. Така ієрархічна організація називається «user-scoped data isolation» (ізоляція даних користувача) і є рекомендованим патерном моделювання в Firestore. Вона дозволяє декларативно обмежити доступ до даних правилами безпеки, що буде показано далі.

2.3.3 Створення композитних індексів. Композитний індекс – це підготовлена структура даних, що оптимізує виконання запиту, який одночасно використовує фільтрацію за одним полем та впорядкування за іншим. У Firestore індекси за окремими полями створюються автоматично для всіх документів, тому декларувати окремо потрібно лише композитні індекси [14].

Для розроблюваної системи визначено два композитні індекси над колекцією events, що оптимізують найбільш частотні запити користувача:

- індекс за полями kind та occurred_at – для побудови сторінки історії з фільтром за типом події (продажі, купівлі, отримання предмета з гри) та сортуванням за часом;
- індекс за полями account_id та occurred_at – для побудови історії подій окремого Steam-акаунта у хронологічному порядку.

Конфігурація композитних індексів зберігається у файлі firestore.indexes.json і застосовується до бази через інструмент командного рядка Firebase CLI. На рис. 8 наведено приклад декларації композитного індексу.

```
{
  "indexes": [
    {
      "collectionGroup": "events",
      "queryScope": "COLLECTION",
      "fields": [
        { "fieldPath": "kind", "order": "ASCENDING" },
        { "fieldPath": "occurredAt", "order": "DESCENDING" }
      ]
    },
    {
      "collectionGroup": "events",
      "queryScope": "COLLECTION",
      "fields": [
        { "fieldPath": "accountId", "order": "ASCENDING" },
        { "fieldPath": "occurredAt", "order": "DESCENDING" }
      ]
    }
  ],
  "fieldOverrides": []
}
```

Рис. 8 Декларація композитного індексу

2.3.4 Створення тригерних функцій. Тригерні функції у Firebase реалізовано через сервіс Cloud Functions. Це фрагменти коду, що автоматично виконуються у відповідь на події бази даних – створення, оновлення або видалення документа [15].

У розроблюваній системі визначено тригерну функцію onAccountDeleted, що реагує на видалення документа з колекції accounts і виконує каскадне видалення всіх предметів, які належали цьому акаунту. Це необхідно для підтримки референційної цілісності даних: оскільки Firestore не контролює зовнішні ключі автоматично, цю логіку реалізовано програмно.

На рис. 9 наведено фрагмент коду тригерної функції.

```
export const onAccountDeleted = onDocumentDeleted(
  'users/{userId}/accounts/{accountId}',
  async (event) => {
    const { userId, accountId } = event.params;
    const db = getFirestore();

    logger.info('Account deleted — cleaning up items', { userId, accountId });

    const snapshot = await db
      .collection(`users/${userId}/items`)
      .where('accountId', '==', accountId)
      .get();

    if (snapshot.empty) {
      logger.info('No items to delete', { userId, accountId });
      return;
    }

    const chunks = chunkArray<QueryDocumentSnapshot>(snapshot.docs, 500);

    for (const chunk of chunks) {
      const batch = db.batch();
      chunk.forEach((doc) => batch.delete(doc.ref));
      await batch.commit();
    }

    logger.info('Items deleted', { userId, accountId, count: snapshot.size });
  }
);
```

Рис. 9 Тригерна функція каскадного видалення предметів акаунта

Функція реалізує таку логіку: при отриманні події видалення документа акаунта вона виконує пошук усіх документів у колекції items, у яких поле account_id збігається з ідентифікатором видаленого акаунта, та видаляє їх

пакетами по 500 документів (відповідно до обмежень Firestore на одну операцію пакетної транзакції).

Решта серверної бізнес-логіки системи (синхронізація інвентарю зі Steam, моніторинг ринкових цін, виставлення предметів на торгових платформах) реалізована не у Cloud Functions, а у окремому сервісі-обробнику на платформі Node.js, розгорнутому на віддаленому сервері. Це свідомий архітектурний вибір: робота зі Steam потребує постійних HTTP-сесій з підтримкою cookies та довгих повторних спроб, що не вкладається у короткоживучу модель безсерверного виконання.

2.3.5 Створення правил безпеки доступу. Розмежування доступу до даних у Firestore реалізовано через декларативну мову Security Rules. Ці правила розгортаються разом з конфігурацією бази та виконуються самим сервісом Firestore перед кожним запитом на читання або запис.

На рис. 10 наведено правила безпеки доступу для розроблюваної системи.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;

      match /accounts/{accountId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }

      match /items/{itemId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }

      match /events/{eventId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }
    }
  }
}
```

Рис. 10 Правила безпеки доступу до Firestore

Логіка правил наступна: користувач може читати або записувати документ лише у тому випадку, коли його ідентифікатор автентифікації (request.auth.uid) збігається з ідентифікатором користувача-власника у шляху

документа. Правила застосовуються рекурсивно до всіх підколекцій: accounts, items, events та strategies. Будь-яка спроба доступу до даних іншого користувача відхиляється на рівні бази, без необхідності перевірки на стороні клієнта чи проміжного сервера.

Такий підхід забезпечує безпеку даних на рівні самого сховища і виключає клас вразливостей, пов'язаних з помилками в логіці контролю доступу у клієнтському застосунку.

2.4 Схема та фізична структура бази даних

Фізична структура бази даних визначає реальну організацію зберігання даних з урахуванням специфіки обраної СУБД. У базі даних розроблюваної системи реалізовано п'ять колекцій документів, ієрархічно організованих під документом користувача.

- users – зберігає кореневі документи користувачів системи. Ідентифікатор документа збігається з ідентифікатором користувача з сервісу Firebase Authentication, що автоматично пов'язує дані з механізмом автентифікації;
- users/{userId}/accounts – підколекція, що зберігає Steam-акаунти, прив'язані до користувача. Кожен документ містить вкладений об'єкт credentials з обліковими даними та вкладений об'єкт platforms з API-ключами і балансами на торгових платформах;
- users/{userId}/items – підколекція з віртуальними предметами усіх Steam-акаунтів користувача. Ідентифікатор документа збігається з ідентифікатором предмета у системі Steam, що гарантує дедуплікацію на рівні бази. Активні виставлення предмета на торгових платформах зберігаються як вкладений масив listings у документі самого предмета;
- users/{userId}/events – підколекція з журналом подій інвентаря (продажі, купівлі, отримання предмета з гри). Ідентифікатори документів

формуються детерміновано з семантичних параметрів події, що забезпечує ідемпотентність повторних записів;

- users/{userId}/strategies – підколекція з правилами автоматичної купівлі недооцінених пропозицій. Кожен документ містить вкладений масив conditions з наборами умов спрацювання (діапазон зношеності, перелік бажаних значень шаблону, максимальна ціна, цільові торгові платформи).

Особливості фізичної структури полягають у трьох ключових моментах.

Вкладені об'єкти замість окремих колекцій

Сутності, що тісно пов'язані з батьківським об'єктом і завжди читаються разом з ним, фізично зберігаються як вкладені об'єкти у документі батьківської сутності. Прикладами такої організації є:

- обліковий запис на торгових платформах (поле platforms у документі акаунта): замість окремої колекції з зовнішнім ключем на акаунт, дані про чотири підтримувані платформи зберігаються як вкладений об'єкт-словник всередині акаунта;
- активні виставлення предмета (поле listings у документі предмета): активні виставлення на різних торгових платформах зберігаються як вкладений масив всередині документа предмета;
- спосіб отримання предмета (поле acquisition): зберігається як вкладений об'єкт з варіативною структурою залежно від способу отримання (отримання з гри, купівля, обмін);
- набори умов спрацювання правила купівлі (поле conditions у документі правила): зберігаються як вкладений масив всередині правила, що дозволяє одному правилу мати декілька альтернативних умов спрацювання.

Такий підхід зменшує кількість окремих запитів до бази при отриманні даних, оскільки усі пов'язані відомості зчитуються однією операцією читання документа.

Денормалізація для забезпечення історичної цілісності

У документі події (events) свідомо дублюються деякі поля предмета (ринкова назва, зображення, тип предмета, гра) – це вкладений об'єкт item_snapshot. Крім того, у подіях продажу зберігається поле cost_basis_usd – кешована ціна купівлі предмета. Дублювання даних є свідомим архітектурним рішенням: воно дозволяє відображати історію подій у читабельному вигляді навіть після видалення відповідного предмета з інвентарю. За класичної реляційної моделі ця інформація отримувалася б через зв'язок із сутністю предмета, але після видалення предмета історія втратила б контекст.

Детерміновані ідентифікатори для ідемпотентності

Ідентифікатори документів колекції events формуються детерміновано з параметрів події за формулами:

- sale-{ідентифікатор_акаунта}-{ідентифікатор_предмета} – для подій продажу;
- purchase-{ідентифікатор_акаунта}-{ідентифікатор_предмета}-{час_у_мілісекундах} – для подій купівлі;
- drop-{ідентифікатор_акаунта}-{ідентифікатор_предмета} – для подій отримання предмета з гри.

Такий підхід гарантує ідемпотентність запису: повторна спроба зберегти ту саму подію не створить дублікат, а оновить вже існуючий документ. Це усуває цілий клас помилок, пов'язаних з паралельним записом одних і тих самих даних з різних компонентів системи.

На рис. 11 наведено приклад реального документа з колекції accounts у консолі Firestore, що ілюструє описану вище структуру – зокрема вкладені поля з даними платформ та облікового запису Steam.

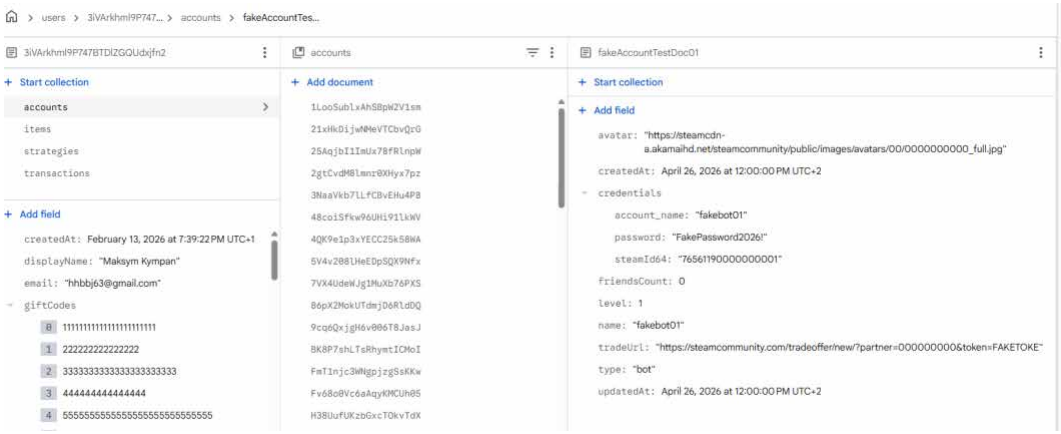

Рис. 11 Приклад документа колекції accounts у консолі Firestore

Отже, фізична структура бази даних поєднує переваги документоорієнтованої моделі (швидкість читання за рахунок вкладеності, гнучкість схеми, інтеграція з механізмом автентифікації) з контрольованою денормалізацією для забезпечення історичної цілісності даних. Такий підхід враховує специфіку предметної області торгівлі віртуальними предметами та забезпечує надійне зберігання інформації з підтримкою механізмів цілісності на рівні самого сховища.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

Перехід від словесного опису предметної області до структурованої моделі починається з виокремлення абстракцій – узагальнених сутностей, які об'єднують типових учасників, об'єкти та процеси досліджуваної сфери діяльності. Для кожної абстракції визначається перелік її важливих властивостей та обов'язків: властивості описують характеристики, які впливають на роботу системи, що проєктується, а обов'язки – дії, що належать до сфери відповідальності цієї сутності [16]. Загальна структура абстракцій, виділених для предметної області торгівлі віртуальними предметами, наведена на рис. 12.

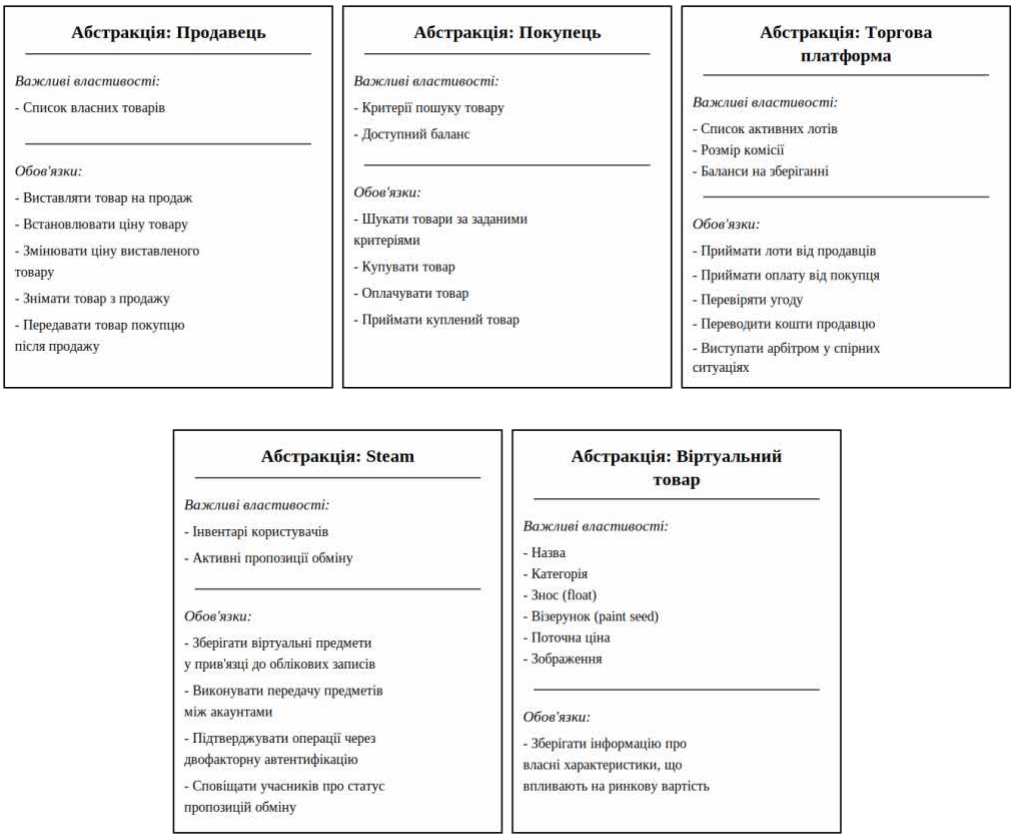

Рис. 12 Абстракції предметної області

У предметній області виокремлено п'ять абстракцій. Перші дві з них – Продавець та Покупець – описують фізичних осіб, які беруть участь у торгових операціях зі своїх облікових записів. Торгова платформа є посередницькою системою, яка приймає від продавців лоти на продаж, утримує оплату покупців на умовному зберіганні та переводить кошти продавцю після того, як угода фактично завершилась. Steam – зовнішня екосистема, у межах якої існують та зберігаються самі віртуальні предмети та через яку відбувається їх безпосередня передача між обліковими записами. Віртуальний товар як абстракція описує об'єкт торгівлі та його характеристики, що формують ринкову вартість.

Відмінною рисою предметної області є чіткий розподіл функцій між торговою платформою та екосистемою Steam. Платформа відповідає за фінансову складову угоди – облік активних лотів, прийом оплати, утримання коштів та розрахунок із продавцем. Steam забезпечує технічну сторону торгівлі – зберігання предметів у прив'язці до облікових записів, виконання передачі через механізм пропозиції обміну та підтвердження операцій засобом двофакторної автентифікації. Такий поділ обов'язків характерний для всіх торгових платформ віртуальних предметів і безпосередньо враховується при подальшому проєктуванні архітектури системи, де модулі взаємодії з торговими платформами та зі Steam реалізуються окремо.

Виокремлені абстракції є основою для побудови діаграми класів та діаграм кооперацій у наступному підрозділі, які описують статичні зв'язки між сутностями та сценарії їх взаємодії у конкретних бізнес-процесах. Атрибути та методи кожного класу безпосередньо випливають із важливих властивостей та обов'язків відповідних абстракцій.

3.2 Моделювання структури та взаємодії об'єктів

Після того, як виокремлено абстракції предметної області, наступним кроком проєктування є моделювання структури та взаємодії об'єктів майбутньої

системи. На цьому етапі визначаються складові системи та зв'язки між ними – не лише статичний перелік сутностей з атрибутами та методами, а й сценарії, у яких ці сутності взаємодіють між собою для реалізації конкретної функціональності [16]. Без такого моделювання перехід до програмного коду супроводжується ризиком реалізувати логіку взаємодії компонентів непослідовно у різних частинах системи.

Для моделювання застосовано стандартну мову UML як загальноприйнятий інструмент графічного представлення об'єктно-орієнтованих моделей. У цьому підрозділі побудовано два типи діаграм, що відображають різні аспекти системи. Статичну структуру описує діаграма класів – вона фіксує перелік сутностей, їх властивості та методи, а також типи зв'язків між ними. Динамічну поведінку описують діаграми кооперацій – вони показують, як об'єкти взаємодіють у межах окремих сценаріїв.

Кожна сутність на діаграмах безпосередньо випливає з відповідної абстракції, описаної у попередньому підрозділі. Властивості абстракції стають атрибутами класу, обов'язки – методами, а характер взаємозв'язків між сутностями визначає тип асоціації, агрегації, композиції чи залежності між класами.

3.2.1 Діаграма класів. Діаграма класів є базовим типом UML-діаграм і призначена для зображення статичної об'єктної структури системи. Вона демонструє основні класи, виявлені на етапі аналізу, їх атрибути, методи та зв'язки між ними. Окрема увага приділяється асоціаціям між класами із зазначенням кратності – такі позначення фіксують, скільки об'єктів одного класу може бути пов'язано з об'єктами іншого класу.

На рис. 13 наведено діаграму класів із асоціаціями. Ця діаграма є основою для подальшого проєктування і реалізації функціональності системи: вона показує, які об'єкти існують, які дані вони зберігають у вигляді атрибутів, які дії можуть виконувати через методи та як ці об'єкти пов'язані між собою.

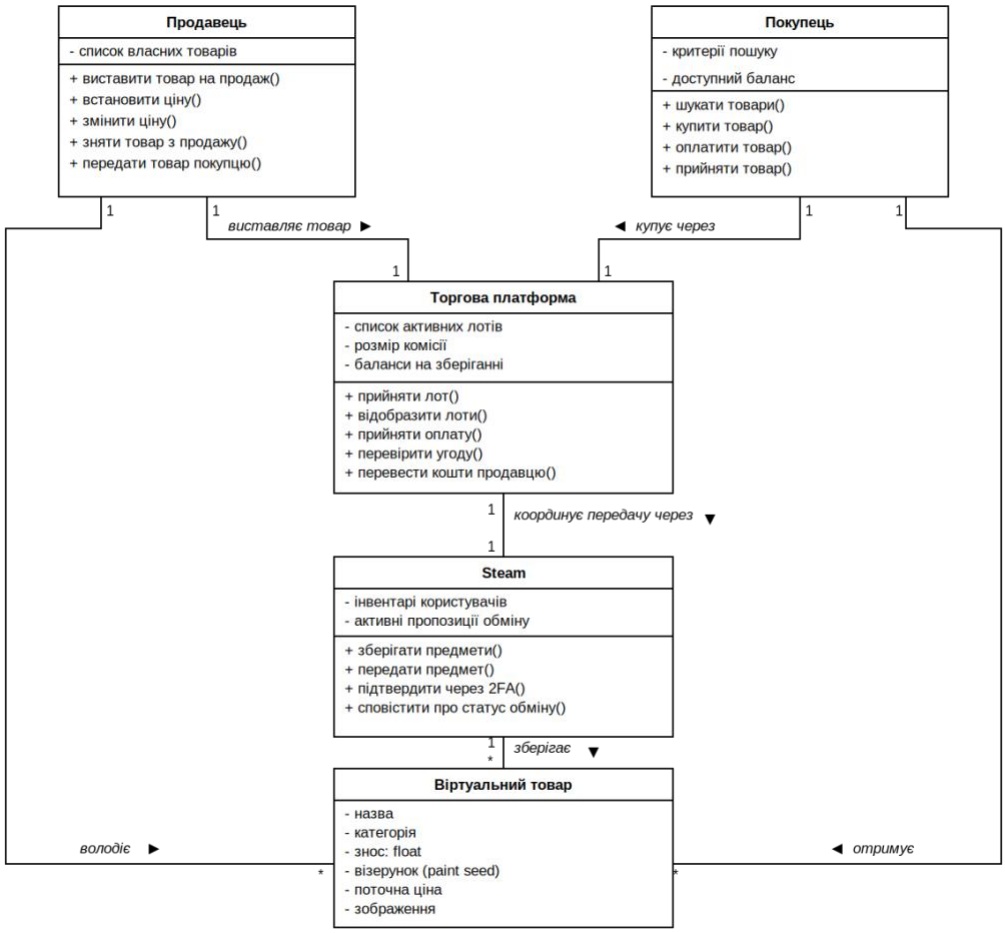

Рис. 13 Діаграма класів із асоціаціями

Діаграма містить п'ять класів, які відповідають виокремленим абстракціям: Продавець, Покупець, Торгова платформа, Steam та Віртуальний товар. Для кожного класу визначено перелік атрибутів, що відображають характеристики відповідної сутності, та операцій, через які ця сутність взаємодіє з іншими учасниками процесу.

Окремий акцент зроблено на типах асоціацій між класами із зазначенням кратності, що дозволяє відобразити природу зв'язків між учасниками предметної області. Продавець виставляє товари на торгову платформу та володіє ними у власному інвентарі – ці відношення реалізовано через дві асоціації типу один до багатьох (1 до 0..*). Покупець купує товари через торгову платформу та після успішної угоди отримує їх у власне володіння; кратність *..1 відображає той факт, що з однією платформою одночасно можуть взаємодіяти багато покупців. Торгова платформа агрегує лоти всіх продавців та

координує передачу предметів через систему Steam, що додатково розкривається у пункті 3.2.2 через окремі діаграми кооперацій. Окремий зв'язок між класами Steam та Віртуальний товар відображає той факт, що віртуальні предмети існують та зберігаються виключно у межах екосистеми Steam.

Побудована діаграма класів узагальнює ключові об'єкти моделі та зв'язки між ними, тим самим формуючи перехід від логічної моделі до її програмного втілення. Подальша деталізація механізмів взаємодії об'єктів подана у наступному пункті через діаграми кооперацій.

3.2.2 Діаграми кооперацій. Діаграми кооперацій є інструментом моделювання динамічних аспектів системи. На відміну від діаграми класів, що описує усю статичну структуру цілісно, кооперація розглядає лише ті класи та зв'язки, які беруть участь у конкретному сценарії. Це дозволяє детально дослідити окремі процеси та визначити, які саме типи відношень – асоціація, агрегація, композиція чи залежність – найкраще описують природу взаємодії об'єктів у кожному випадку [16].

Для системи виокремлено чотири прості кооперації. Усі вони побудовані на основі тих самих п'яти класів, що подано на загальній діаграмі класів, без введення нових сутностей.

Кооперація «Виставлення товару на продаж». У цьому сценарії продавець публікує власний віртуальний товар на торговій платформі. Кооперація задіює три класи: Продавець, Торгова платформа та Віртуальний товар. Зв'язок між класами Продавець і Торгова платформа є асоціацією, оскільки продавець ініціює взаємодію з платформою для публікації лотів. Зв'язок між класами Продавець і Віртуальний товар – асоціація типу один до багатьох: продавець володіє переліком власних товарів, проте товари існують незалежно від нього у межах системи Steam. Кооперація наведена на рис. 14.

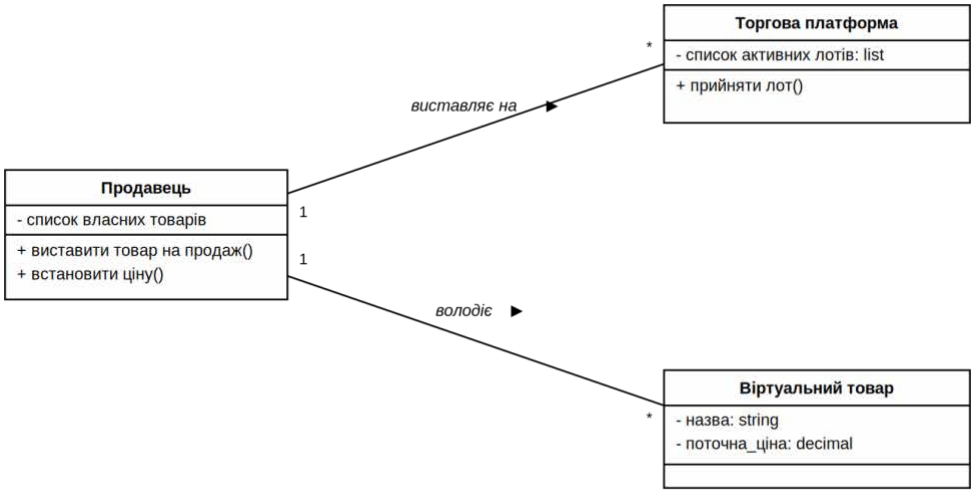

Рис. 14 Діаграма кооперації «Виставлення товару на продаж»

Кооперація «Утримання активних лотів». У цьому сценарії торгова платформа підтримує список активних лотів, кожен з яких відповідає виставленому віртуальному товару. Задіяні два класи: Торгова платформа та Віртуальний товар. Зв'язок між ними є агрегацією: платформа об'єднує товари у власний список активних лотів, але не контролює їх повністю. Якщо платформа припинить роботу, віртуальні товари не зникнуть, а залишаться в інвентарях користувачів у Steam. Кооперація наведена на рис. 15.

Рис. 15 Діаграма кооперації «Утримання активних лотів»

Кооперація «Передача предмета між акаунтами». У цьому сценарії торгова платформа ініціює передачу віртуального товару через систему Steam, використовуючи її програмний інтерфейс. Кооперація задіює три класи – Торгова платформа, Steam та Віртуальний товар – і демонструє одночасно два типи зв'язку. Між класами Торгова платформа і Steam – залежність: платформа

використовує функціонал Steam для виконання передачі, але не володіє ним і не є його частиною. Між класами Steam і Віртуальний товар – композиція: віртуальні товари існують виключно у межах екосистеми Steam, і у разі припинення існування облікового запису Steam пов'язані з ним предмети також перестають існувати. Кооперація наведена на рис. 16.

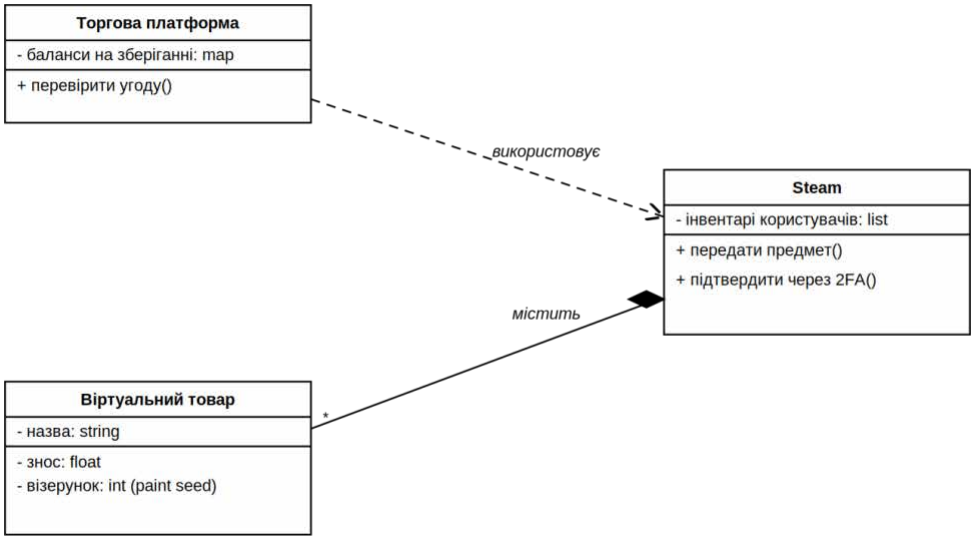

Рис. 16 Діаграма кооперації «Передача предмета між акаунтами»

Кооперація «Придбання товару». У цьому сценарії покупець здійснює пошук та придбання віртуального товару через торгову платформу. Задіяні три класи: Покупець, Торгова платформа та Віртуальний товар. Зв'язок між класами Покупець і Торгова платформа є асоціацією, оскільки покупець ініціює взаємодію з платформою для пошуку та купівлі товарів; її кратність – багато до одного – відображає, що множина покупців може одночасно користуватися однією платформою. Зв'язок між класами Покупець і Віртуальний товар – також асоціація, оскільки покупець отримує придбані товари у власне володіння після успішної угоди. Кооперація наведена на рис. 17.

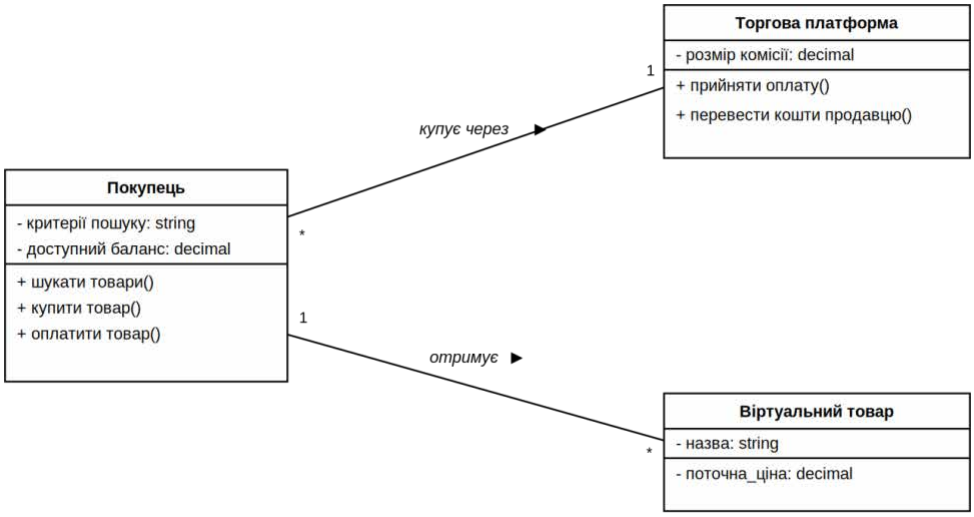

Рис. 17 Діаграма кооперації «Придбання товару»

Описані кооперації деталізують механіку основних сценаріїв системи й слугують основою для подальшого проєктування організаційної структури програмного забезпечення, яка розглядається у наступному підрозділі.

3.3 Організаційна структура програмного забезпечення

Для високорівневого представлення архітектури використано діаграму пакетів – вона показує, як функціональність системи групується у логічні модулі з власною зоною відповідальності та які залежності існують між цими модулями [16]. Діаграма пакетів забезпечує високорівневе уявлення про архітектуру системи – вона показує, як саме функціональність розподілена між клієнтською та серверною частинами, які з пакетів є зовнішньо доступними, а які виконують виключно внутрішні службові функції. Діаграма пакетів програмного забезпечення розміщена в Додатку Д.

Усі пакети системи об'єднано у два верхньорівневих контейнери: Client та Server, що відображає клієнт-серверну архітектуру. Кожен контейнер містить пакети з власною зоною відповідальності, а зв'язки між пакетами реалізовано через залежності, які показують напрямок використання функціональності.

Опис пакетів

UserInterface – пакет графічного інтерфейсу клієнтської частини, побудований на основі SvelteKit-сторінок та компонентів Shadcn-svelte. Він відповідає за візуальне представлення даних користувачеві – відображення інвентарю, керування обліковими записами Steam, налаштування правил автоматичної купівлі та перегляд історії торгових операцій. Пакет не містить бізнес-логіки і взаємодіє з іншими модулями виключно через пакет ClientCore.

ClientCore – ядро клієнтської частини, яке інкапсулює сервісний шар для роботи з даними. У ньому реалізовані функції автентифікації користувача, читання та запису документів у базу даних, локальне кешування інвентарю та зв'язок із серверною частиною. ClientCore залежить від двох пакетів серверної частини: Authentication – для отримання токенів доступу, та DataStorage – для безпосередньої роботи з документами користувача.

Authentication – пакет автентифікації, який реалізовано на основі сервісу Firebase Authentication з підтримкою Google OAuth. Він видає JWT-токени, що використовуються для подальших звернень до інших серверних пакетів. Authentication залежить від DataStorage – для збереження та читання документа користувача, який створюється при першій реєстрації.

TradingEngine – основний серверний пакет із бізнес-логікою торгівлі віртуальними предметами. Він відповідає за роботу зі Steam-сесіями облікових записів користувача, формування та підтвердження пропозицій обміну, аналіз ринкових пропозицій та автоматичне виявлення недооцінених лотів за активними правилами купівлі. TradingEngine має три залежності: від Authentication – для перевірки прав доступу, від DataStorage – для збереження результатів торгових операцій та зчитування активних правил, та від PlatformIntegrations – для отримання даних із зовнішніх торгових платформ.

PlatformIntegrations – пакет інтеграції зі сторонніми торговими платформами. Він містить клієнти для роботи з програмними інтерфейсами Steam Community Market, CSFloat, Market.CSGO та Youpin – чотирьох платформ, з якими взаємодіє система. PlatformIntegrations нормалізує дані з різних джерел до єдиного внутрішнього формату та надає TradingEngine

уніфікований доступ до ринкової інформації незалежно від платформи. Залежить від Utils для допоміжних функцій.

DataStorage – пакет роботи з базою даних, побудований над Firestore. Він надає типізований доступ до колекцій users, accounts, items, events та strategies, які зберігають дані користувачів, прив'язані Steam-акаунти, віртуальні предмети, історію подій та правила автоматичної купівлі. DataStorage є центральним пакетом серверної частини – до нього звертаються Authentication, TradingEngine, а також ClientCore безпосередньо з клієнтської частини. Сам же DataStorage залежить лише від Utils.

Utils – внутрішній службовий пакет, який містить допоміжні функції: шифрування конфіденційних даних облікових записів Steam, валідацію даних за допомогою Zod-схем, обчислення дат закінчення обмежень торгових операцій та перерахунок цін між валютами торгових платформ. На відміну від інших пакетів, Utils має модифікатор приватного доступу і призначений для використання виключно іншими серверними пакетами; зовні системи його функції недоступні.

Залежності між пакетами

Стрілки на діаграмі позначають напрям використання функціональності одного пакета іншим. Аналіз залежностей дозволяє виявити кілька важливих особливостей архітектури.

Перш за все, клієнтська частина не має прямого доступу до пакета TradingEngine: ClientCore звертається до серверної частини лише через Authentication та DataStorage. Це є наслідком архітектурного вибору, у якому користувач не запускає торгові операції на запит, а лише налаштовує правила та переглядає результати. Сам TradingEngine працює як автономний серверний сервіс, який реагує на зміни у базі даних: коли користувач створює нове правило купівлі, TradingEngine виявляє його, починає моніторинг ринкових пропозицій та виконує купівлю при виявленні відповідного лоту.

Друга особливість – центральна роль пакета DataStorage. До нього звертаються чотири з шести інших пакетів. Така схема відповідає одному з

основних патернів роботи з Firestore: обмін даними між компонентами системи відбувається не через прямі виклики між компонентами, а через спільну базу даних, до якої компоненти отримують доступ із застосуванням однакових правил безпеки.

Третя особливість – виокремлення пакета Utils як внутрішнього з модифікатором приватного доступу. Допоміжні функції шифрування, валідації та конверсії використовуються у багатьох місцях серверної частини, але не повинні бути доступними ззовні. Виділення цих функцій у окремий приватний пакет дозволяє повторно використовувати їх без дублювання коду та водночас зберігати чіткі межі публічного API серверної частини.

Описана організаційна структура утворює модульну архітектуру, у якій кожен пакет має чітко окреслену зону відповідальності та обмежений перелік залежностей. Це спрощує розробку, тестування та супровід окремих частин системи без впливу на інші пакети.

3.4 Компонентна структура системи

Компонентна структура програмного забезпечення представлена у вигляді діаграми компонентів. Вона моделює фізичну архітектуру системи – перелік виконуваних середовищ, бібліотек, вихідних файлів та зовнішніх сервісів, а також зв'язки між ними [16]. На відміну від діаграми пакетів, яка фіксує логічний поділ функціональності, діаграма компонентів зображує систему такою, якою вона існує під час розгортання та виконання. Діаграма компонентів програмного забезпечення розміщена в Додатку Е.

Діаграма побудована з використанням стандартних стереотипів UML 2.x: «executable» – виконуване середовище, «source» – вихідний файл, «component» – програмний компонент, «library» – скомпільована бібліотека, «service» – зовнішній сервіс, «table» – колекція бази даних. Усі компоненти системи розподілено між п'ятьма контейнерами, які відповідають фізично відокремленим середовищам виконання. Такий поділ відповідає принципам

побудови розподілених систем, де незалежні обчислювальні вузли взаємодіють через мережу та сприймаються користувачем як єдина узгоджена система [17].

Клієнтська частина (Client Side)

Контейнер клієнтської частини містить три виконувані середовища, що поділяють спільну кодову базу SvelteKit-додатку. Перше – вебзастосунок, який запускається у браузері користувача за доменом, на якому розгорнуто систему. Друге – десктопний застосунок на основі Tauri 2.0, що пакується у нативні виконувані файли для Windows, macOS та Linux. Третє – мобільний застосунок на тій же платформі Tauri, що постачається у вигляді .apk-пакета для Android. Усі три середовища отримуються з єдиної статичної збірки SvelteKit через адаптер @sveltejs/adapter-static, що дозволяє підтримувати спільну кодову базу без дублювання логіки.

Спільна бібліотека SvelteKit-фронтенду поділена на три шари вихідного коду: routes/ – каталог сторінок з файлами +page.svelte та +layout.svelte; components/ui/ – повторно використовувані UI-компоненти на основі Shadcn-svelte; lib/services/ – сервісний шар з модулями firestore.service, accounts.service, inventory.service, trade.service та listing.service, що інкапсулюють доступ до бази даних та взаємодію зі серверною частиною. Стан автентифікації користувача зберігається у Svelte 5 runes-store auth.svelte.ts, а Zod-схеми для валідації даних – у спільному каталозі shared/types/.

Крайова мережа та тунель (Edge / Tunnel)

Доступ до серверних сервісів забезпечується через Cloudflare Tunnel без відкритих публічних портів на сервері. Контейнер містить два компоненти: Cloudflare Edge – глобальну мережу Cloudflare, яка виконує функції DNS, веб-фаєрволу та термінації TLS на порту 443; cloudflared – клієнт тунелю, що встановлює зашифроване вихідне з'єднання з мережею Cloudflare. Конфігурація тунелю зберігається у файлах config.yml та tunnel-credentials.json.

Серверна частина (Worker Tier)

Серверна частина реалізована у вигляді Express Worker – основного серверного застосунку із бізнес-логікою торгівлі, що працює як довгоживучий

процес на платформі Node.js та підтримує постійні з'єднання зі Steam Network. Express Worker структурно складається з чотирьох програмних компонентів. SteamModule – модуль роботи зі Steam через бібліотеки steam-user, steam-totp та steamcommunity; підтримує сесії облікових записів, оновлення токенів та авторизацію через генерацію кодів двофакторної автентифікації. TradeModule – модуль торгових операцій на основі бібліотек steam-tradeoffer-manager та globaloffensive; виконує надсилання та підтвердження пропозицій обміну, а також інспекцію предметів за параметрами зношеності та шаблону. MarketplaceModule – модуль інтеграції з третіми торговими платформами; містить клієнти csfloat.client, marketcsgo.client та youpin.client для отримання ринкових даних з відповідних платформ. PricingModule – модуль ціноутворення та валютної конверсії, який агрегує курси з декількох джерел та нормалізує ціни до єдиної валюти для порівняння пропозицій з різних платформ.

Точкою входу Express Worker є файли main.ts та app.ts, які імпортують усі чотири модулі. HTTP-маршрути реалізовано у каталозі routes/ (steam-login.ts, trade-send.ts), а сервіси бізнес-логіки – у services/ (steam-session.ts, trade-manager.ts).

Хмарна інфраструктура (Firebase Cloud)

Контейнер Firebase Cloud містить три сервіси одного Firebase-проєкту. Firebase Authentication виконує автентифікацію користувачів через Google OAuth та видає JWT-токени для подальших звернень до інших серверних компонентів. Cloud Functions – безсерверні функції, розгорнуті на Google Cloud Platform; вихідний код знаходиться у functions/src/index.ts, скомпільований артефакт – у functions/lib/index.js. Поточна реалізація містить тригер onAccountDeleted, який каскадно видаляє пов'язані документи items після видалення документа облікового запису. Firestore Database – документоорієнтована база даних із п'ятьма колекціями users, accounts, items, strategies, transactions; правила безпеки задані у файлі firestore.rules.

Розподіл операцій запису між клієнтом та серверною частиною відбувається у такий спосіб: клієнт пише до Firestore через Firebase Web SDK

лише легкі дані – профіль користувача та метадані облікових записів; Express Worker пише до Firestore через Firebase Admin SDK великі обсяги – партії items та історію transactions. Для останніх правило allow write: if false у firestore.rules виключає можливість їх запису з клієнта.

Зовнішні сервіси (External APIs)

Контейнер зовнішніх сервісів містить чотири торгові платформи, з якими взаємодіє система. Steam Network та Steam Web API – основна платформа, з якою серверна частина працює одночасно через два протоколи: бінарний TCP через бібліотеку steam-user (для роботи з ігровим клієнтом та інспекції предметів) та HTTPS REST через Steam Web API (для отримання даних інвентарю та маркетплейсу). CSFloat API, Market.CSGO API та Youpin API – сторонні торгові платформи, з якими система обмінюється даними про лістинги через HTTPS REST.

Описана компонентна структура реалізує розподілену архітектуру, у якій кожне середовище виконання має чітко окреслену зону відповідальності. Клієнтська частина зосереджена на представленні даних користувачу, серверна частина – на виконанні торгових операцій та інтеграції з зовнішніми сервісами, хмарна інфраструктура Firebase – на автентифікації, зберіганні даних та виконанні фонових тригерів. Така архітектура забезпечує модульність системи: окремі компоненти можуть оновлюватися або масштабуватися незалежно від інших.

3.5 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка програмного забезпечення здійснювалась з використанням сучасного інструментарію, що поєднує єдину кодову базу для клієнтських застосунків у трьох форматах поставки, документоорієнтовану хмарну базу даних та спеціалізовані бібліотеки для взаємодії з ігровою платформою Steam. Як основну мову програмування обрано TypeScript, що забезпечує єдиний

синтаксис клієнтської та серверної частин і дозволяє повторно використовувати спільні типи даних. Розробка ведеться у середовищі Visual Studio Code, управління залежностями – через пакетний менеджер npm.

Клієнтська частина.

Клієнтський застосунок побудовано на фреймворку SvelteKit 5.0 з бібліотекою UI-компонентів Shadcn-svelte та стилізацією через TailwindCSS. Для пакування у нативні застосунки для Windows, macOS, Linux та Android використано платформу Tauri 2.0, що дозволяє з єдиної кодової бази формувати десктопну, мобільну та вебверсії системи. На відміну від Electron, Tauri використовує системний WebView, що значно зменшує розмір збірки та споживання пам'яті.

Серверна частина.

Серверну частину реалізовано на платформі Node.js з HTTP-фреймворком Express. Робота зі Steam Network виконується через спеціалізовані бібліотеки steam-user, steam-totp, steamcommunity, steam-tradeoffer-manager та globaloffensive, що забезпечують повноцінну взаємодію з ігровою платформою без офіційних обмежень Steam Web API – підключення до Steam Network через бінарний TCP-протокол, генерацію одноразових TOTP-кодів двофакторної автентифікації, керування пропозиціями обміну та інспекцію предметів для отримання значень float і paint seed. Інтеграція зі сторонніми торговими платформами CSFloat, Market.CSGO та Youpin реалізована окремими HTTP-клієнтами, що нормалізують ринкові дані до єдиного внутрішнього формату.

Інфраструктура та допоміжні засоби.

Зберігання даних виконується у документоорієнтованій базі Cloud Firestore у складі екосистеми Firebase, що додатково надає сервіс автентифікації Firebase Authentication через Google OAuth та засіб виконання тригерних функцій Cloud Functions. У клієнтській частині використано Firebase Web SDK, у серверній – Firebase Admin SDK з повним адміністративним доступом. Валідація даних на межі клієнт-сервер виконується бібліотекою Zod, конфіденційні дані облікових записів Steam шифруються алгоритмом AES-256-

GCM, зовнішній доступ до серверної частини забезпечується через Cloudflare Tunnel без відкриття публічних портів на сервері.

3.6 Алгоритмізація та програмування програмних модулів

Перш ніж перейти до програмного коду, необхідно зафіксувати алгоритмічну сторону ключових бізнес-процесів: послідовність кроків, умови переходу між ними та реакцію системи на типові помилки. Окремі алгоритми супроводжуються блок-схемами, на яких графічно зображено потік керування, та фрагментами вихідного коду, які демонструють особливості програмної реалізації.

У межах цього підрозділу детально розглянуто алгоритм імпорту облікових записів Steam – одну з центральних функцій системи, через яку користувач передає до системи дані своїх торгових облікових записів разом із секретами двофакторної автентифікації. Цей алгоритм обрано для демонстрації тому, що він поєднує одночасно декілька типових аспектів реалізації: парсинг файлів на клієнтській стороні, валідацію даних з використанням спільної Zod-схеми, автентифікацію HTTP-запитів через Firebase ID Token, інтеграцію зі Steam Network засобами спеціалізованих бібліотек, шифрування конфіденційних даних та запис у документоорієнтовану базу даних.

На рис. 18 наведена блок-схема алгоритму імпорту облікових записів Steam.

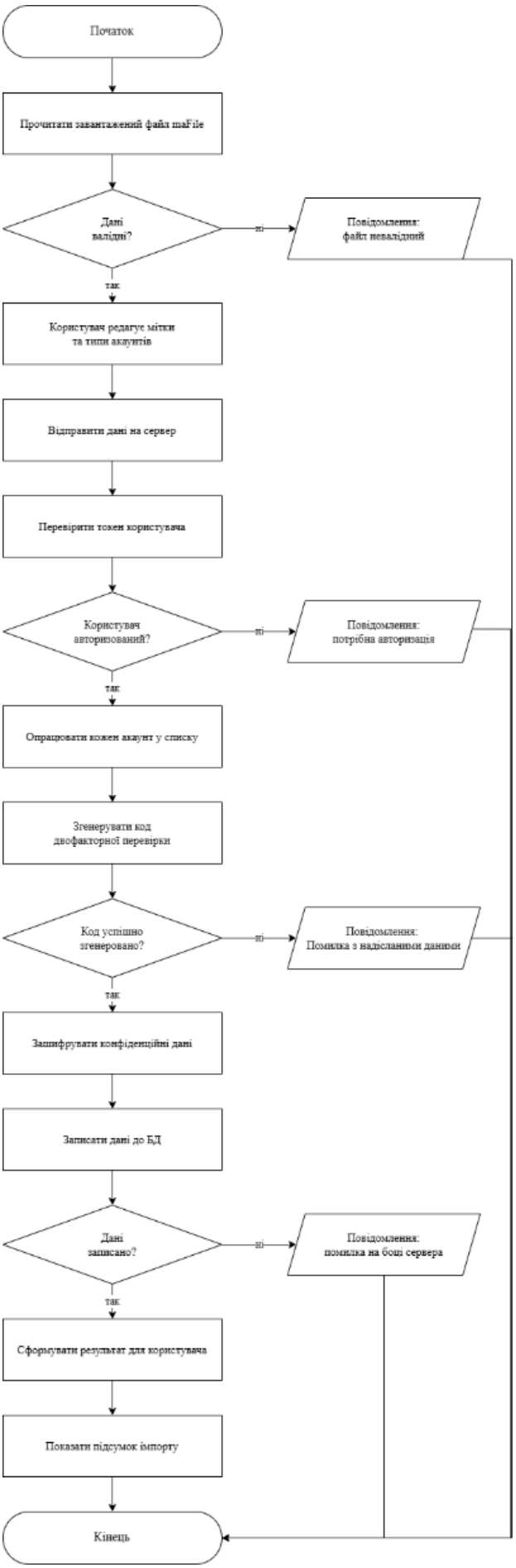

Рис. 18 Блок-схема алгоритму імпорту облікових записів Steam

Процес починається з того, що користувач завантажує один або декілька файлів .maFile, які містять секрети двофакторної автентифікації облікових записів Steam, експортовані з мобільного застосунку Steam Mobile Authenticator. Клієнтська частина системи зчитує вміст кожного файлу та валідує його структуру з використанням Zod-схеми. У разі невдалої валідації файл позначається статусом помилки, а користувачеві виводиться відповідне повідомлення.

Далі користувач у покроковій формі вводить додаткові параметри облікових записів – пароль до кожного з них (у форматі login:password для зручності масового імпорту з текстового файлу), а також обирає тип акаунту (основний акаунт, акаунт-сховище або акаунт-бот). Сформований масив облікових записів надсилається на серверну частину одним HTTP-запитом у форматі JSON, при цьому в заголовку Authorization передається Firebase ID Token поточного користувача.

На стороні серверної частини запит послідовно проходить два рівні middleware. Перший – authenticate – перевіряє валідність ID Token через Firebase Admin SDK та додає у запит ідентифікатор користувача. Другий – validateRequest – повторно валідує тіло запиту Zod-схемою, що відсікає некоректні дані ще до потрапляння в основну логіку. У разі помилки на будь-якому з рівнів сервер повертає відповідне повідомлення, і обробка завершується.

Основна обробка запиту в контролері реалізована як послідовність чотирьох незалежних фаз, що дозволяє ізолювати помилки кожної фази та формувати детальну звітність про результат імпорту. Перша фаза – фільтрація вже існуючих облікових записів у користувача шляхом пошуку у Firestore за іменами акаунтів. Друга фаза – автентифікація у Steam Network для кожного нового облікового запису за допомогою бібліотек steam-user та steam-totp. Третя фаза – шифрування конфіденційних даних та збереження документа облікового запису у Firestore через Admin SDK. Четверта фаза – імпорт інвентарю облікового запису з використанням cookies, отриманих під час логіну. Після

завершення усіх фаз формується підсумкова відповідь з агрегованими результатами та лічильниками успішних і неуспішних операцій.

Структура файлу .maFile описана Zod-схемою, що використовується як на клієнтській, так і на серверній частинах. Базова схема maFileSchema розширюється для опису проміжного стану під час покрокової форми та фінального стану перед надсиланням на сервер. Ієрархія схем наведена на рис. 19.

```
import { z } from 'zod';

const importStatusSchema = z.enum(['valid', 'error']);

export const accountTypeSchema = z.enum(['main', 'storage', 'bot']);
export type AccountType = z.infer<typeof accountTypeSchema>;

export const maFileSchema = z.object({
    account_name: z.string().min(1, 'Account name is required'),
    shared_secret: z.string(),
    identity_secret: z.string(),
    revocation_code: z.string().length(6, 'Revocation code must be exactly 6 characters')
});

export const importedAccountSchema = maFileSchema.extend({
    password: z.string().optional(),
    type: accountTypeSchema.default('bot'),
    status: importStatusSchema,
    error: z.string().optional()
});

export type ImportedAccount = z.infer<typeof importedAccountSchema>;

export const accountForImportSchema = importedAccountSchema.required({ password: true });
export type AccountForImport = z.infer<typeof accountForImportSchema>;
```

Рис. 19 Ієрархія Zod-схем для валідації даних імпорту

Базова схема maFileSchema описує чотири обов'язкових поля, що містяться у файлі .maFile: account_name – ім'я облікового запису Steam, shared_secret – секрет для генерації одноразових кодів двофакторної автентифікації, identity_secret – секрет для підтвердження пропозицій обміну, revocation_code – резервний код відновлення доступу. Через метод extend схема розширюється до importedAccountSchema, що описує тимчасовий стан облікового запису під час проходження покрокової форми, а фінальна схема accountForImportSchema через метод required робить поле password

обов'язковим. Така триступенева ієрархія дозволяє коректно типізувати дані на кожному етапі обробки без дублювання описів полів.

На стороні серверної частини запит послідовно проходить два рівні middleware: authenticate – перевіряє Firebase ID Token через Firebase Admin SDK та додає до запиту ідентифікатор користувача; validateRequest – повторно валідує тіло запиту Zod-схемою. У разі помилки на будь-якому з рівнів сервер повертає відповідне повідомлення, і обробка завершується. Основну логіку обробки запиту реалізовано в контролері importAccounts як послідовність чотирьох незалежних фаз – її фрагмент наведено на рис. 20.

```
async importAccounts(
    req: Request<{}, {}, AccountForImport[]>,
    res: Response<BatchImportResponse>
): Promise<void> {
    const accounts = req.body;
    const userId = req.userId;

    logger.info('Starting batch account import', {
        count: accounts.length,
        userId
    });

    const accountNames = accounts.map((a) => a.account_name);
    const { existing, newAccounts: newAccountNames } =
        await this.firestoreService.filterExistingAccounts(userId, accountNames);

    const earlyErrors = existing.map((name) => ({
        error: `Account "${name}" already exists`,
        accountName: name
    }));

    if (earlyErrors.length > 0) {
        logger.info('Skipping already-existing accounts before Steam login', {
            userId,
            skipped: existing
        });
    }

    const accountsToProcess = accounts.filter((a) => newAccountNames.includes(a.account_name));

    const steamResults = await this.steamService.processAccountsBatch(accountsToProcess);

    const successfulAccounts = steamResults
        .filter((result): result is { account: Account; cookies: string[]; steamClient: SteamUser; accountName: string } => 'account' in result)
        .map((result) => result.account);
```

Рис. 20 Контролер імпорту облікових записів з чотирифазною логікою

Контролер importAccounts реалізує послідовність із чотирьох фаз: фільтрація вже існуючих акаунтів через Firestore, паралельна автентифікація у Steam Network з генерацією TOTP-кодів засобами бібліотеки steam-totp, шифрування конфіденційних полів алгоритмом AES-256-GCM та збереження документів облікових записів у Firestore, після чого виконується пакетний імпорт інвентарю кожного облікового запису з використанням cookies сесії Steam. Після завершення усіх фаз результати агрегуються у єдиний масив,

обчислюються лічильники successful та failed, і клієнтській частині повертається підсумкова відповідь у форматі BatchImportResponse.

Деталі функції шифрування encryptCredentials, що використовується у третій фазі, наведено на рис. 21.

```
import { createCipheriv, createDecipheriv, randomBytes } from 'crypto';
import { AppErrors } from '../utils/appError';

export interface EncryptedCredentials {
    ciphertext: string;
    iv: string;
    authTag: string;
}

export function encryptCredentials(plaintext: string): EncryptedCredentials {
    const key = Buffer.from(process.env.ENCRYPTION_KEY!, 'hex');
    if (key.length !== 32) throw AppErrors.internalServerError('ENCRYPTION_KEY must be 32 bytes');

    const iv = randomBytes(12);
    const cipher = createCipheriv('aes-256-gcm', key, iv);
    const encrypted = Buffer.concat([cipher.update(plaintext, 'utf8'), cipher.final()]);

    return {
        ciphertext: encrypted.toString('base64'),
        iv: iv.toString('base64'),
        authTag: cipher.getAuthTag().toString('base64')
    };
}

export function decryptCredentials(encrypted: EncryptedCredentials): string {
    const key = Buffer.from(process.env.ENCRYPTION_KEY!, 'hex');
    const iv = Buffer.from(encrypted.iv, 'base64');
    const ciphertext = Buffer.from(encrypted.ciphertext, 'base64');

    const decipher = createDecipheriv('aes-256-gcm', key, iv);
    decipher.setAuthTag(Buffer.from(encrypted.authTag, 'base64'));

    return Buffer.concat([decipher.update(ciphertext), decipher.final()]).toString('utf8');
}
```

Рис. 21 Функція шифрування конфіденційних даних облікового запису

Функція encryptCredentials приймає JSON-серіалізовані конфіденційні поля облікового запису та повертає об'єкт із трьома закодованими у Base64 полями: ciphertext – власне шифротекст; iv – випадковий 12-байтовий вектор ініціалізації, що гарантує унікальність кожної операції шифрування; authTag – тег автентичності, який дозволяє виявити будь-яку модифікацію шифротексту при подальшому розшифруванні. Розмір вектора ініціалізації та тегу автентичності відповідають параметрам режиму Galois/Counter Mode, специфікованого у стандарті NIST SP 800-38D [18]. 256-бітний ключ шифрування зчитується з налаштувань середовища виконання Worker-сервісу

та ніколи не передається у відкритому вигляді через мережу і не зберігається у вихідному коді.

Описана програмна реалізація демонструє декілька важливих архітектурних рішень. Спільна Zod-схема maFileSchema повторно використовується між клієнтською та серверною частинами через пакет shared/types, що усуває дублювання валідаційного коду. Послідовне застосування middleware authenticate і validateRequest на рівні маршрутизатора відокремлює технічні аспекти HTTP-обробки від бізнес-логіки контролера. Розділення основної логіки на чотири незалежні фази дозволяє ізолювати помилки кожної фази та формувати детальну звітність про результат імпорту, навіть якщо одна з фаз для конкретного облікового запису завершилася невдало.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Перед передачею розробленої системи до експлуатації було виконано комплексну перевірку її функціональності з метою підтвердження стійкості роботи, повноти реалізації заявлених можливостей та відсутності критичних дефектів. Перевірка охоплювала як окремі програмні модулі (шифрування, валідаційні схеми, генерація кодів двофакторної автентифікації), так і цілісну поведінку розподіленої системи: взаємодію клієнтських застосунків через Cloudflare Tunnel із серверним обробником, обмін даними з ігровою платформою Steam і сторонніми торговими майданчиками, синхронізацію документів у Firestore та реакцію автоматизованих правил купівлі на появу недооцінених лотів.

У межах перевірки було застосовано декілька взаємодоповнюючих підходів, кожен з яких відповідає за свій рівень контролю якості. Класифікація видів тестування програмного забезпечення на модульне, інтеграційне, системне та приймальне є усталеною практикою, описаною у класичній літературі з програмної інженерії [19, 20].

- Модульне тестування (Unit Testing) – охоплювало ізольовану перевірку окремих програмних блоків: функцій шифрування облікових даних алгоритмом AES-256-GCM, Zod-схем валідації документів колекцій Firestore, генерації одноразових TOTP-кодів за shared_secret, обчислення комісій торгових платформ та формування детермінованих ідентифікаторів подій інвентаря. Кожен блок перевірявся незалежно від решти системи на наборі заздалегідь визначених вхідних даних з очікуваним результатом;
- Системне тестування (System Testing) – здійснювалося над системою у цілому в умовах, наближених до реальної експлуатації: усі компоненти

розгорталися у штатних середовищах (Docker-контейнери серверної частини на VPS, бінарні збірки клієнтів для трьох типів пристроїв, окремий проєкт Firebase). Перевірялася здатність системи коректно опрацьовувати тривалі сценарії: одночасну роботу з декількома Steam-сесіями, неперервний моніторинг ринкових пропозицій та опрацювання тимчасових збоїв програмних інтерфейсів зовнішніх сервісів;

- Тестування «чорної скриньки» (Black Box Testing) – передбачало перевірку поведінки системи з точки зору кінцевого користувача, який не має жодного уявлення про внутрішню реалізацію. Тестувальник виконував завдання за наперед визначеним сценарієм та фіксував відповідність фактичного результату очікуваному [21].

Далі наведено детальний приклад перевірки методом «чорної скриньки», що ілюструє повний цикл взаємодії користувача з розробленою системою – від моменту входу в обліковий запис до виставлення віртуального предмета на продаж. Цей сценарій було обрано як такий, що задіює всі ключові підсистеми та дозволяє одночасно перевірити цілісність роботи клієнтської частини, серверного обробника, ігрової платформи Steam та сторонніх торгових майданчиків.

Приклад тестування за методом «чорної скриньки»

Наведений сценарій імітує типову роботу учасника ринку віртуальних предметів, який вперше отримав доступ до системи та виконує послідовність основних операцій. Кожен крок супроводжується фіксацією результату та порівнянням з очікуваним станом інтерфейсу.

1. Вхід у систему через обліковий запис Google

Користувач відкриває застосунок та натискає кнопку входу через Google. Після OAuth-діалогу сервіс Firebase Authentication видає JWT-токен сесії, а система перенаправляє користувача до головного екрана (рис. 22).

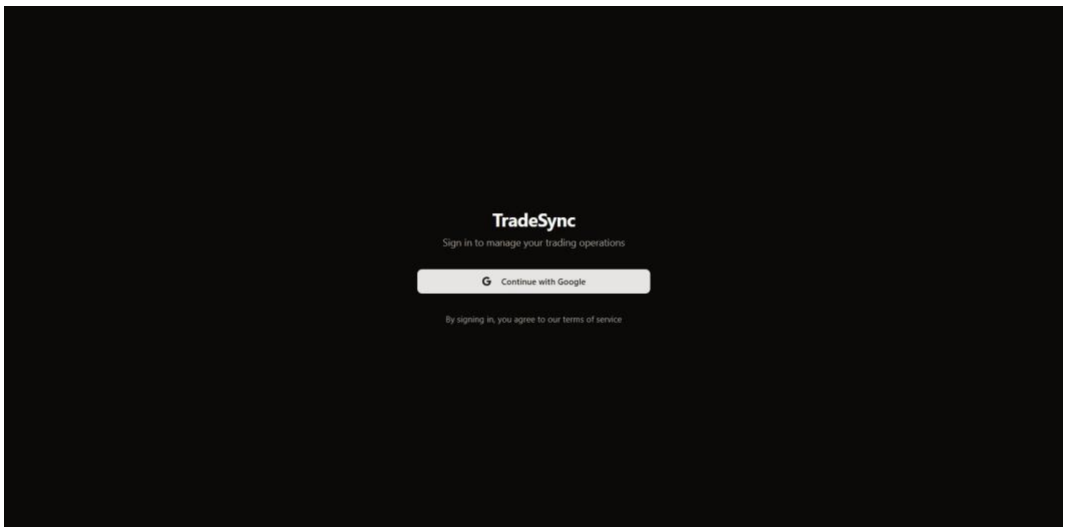

Рис. 22 Сторінка входу до системи через обліковий запис Google

2. Перевірка головного екрана

Після авторизації відкривається сторінка Dashboard з чотирма KPI-плитками (Net P&L, Items sold, Avg margin, Listed value), стовпчиковим графіком прибутковості за обраний період та блоком останньої активності. Перемикач у правому верхньому куті дозволяє вибрати діапазон 7d / 30d / 90d / all. Головний екран наведено на рис. 23.

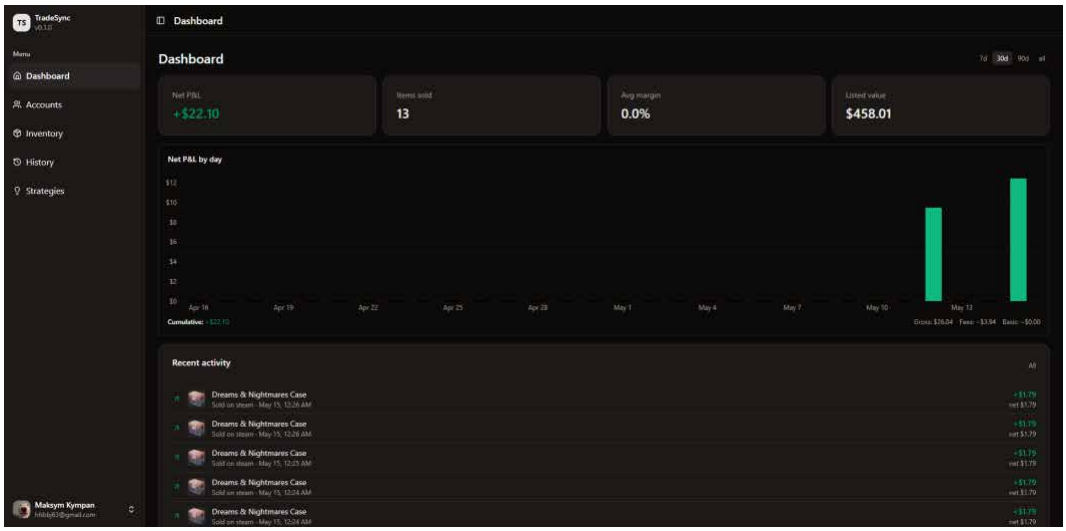

Рис. 23 Головний екран системи після авторизації користувача

3. Імпорт облікових записів Steam

Майстер імпорту реалізовано у вигляді покрокової форми з трьома етапами – «Upload Files», «Review & Edit» та «Confirm». На першому кроці користувач перетягує файли двофакторної автентифікації .maFile у виділену

область або обирає їх через діалог браузера (рис. 24); клієнтська частина валідує кожен файл за Zod-схемою.

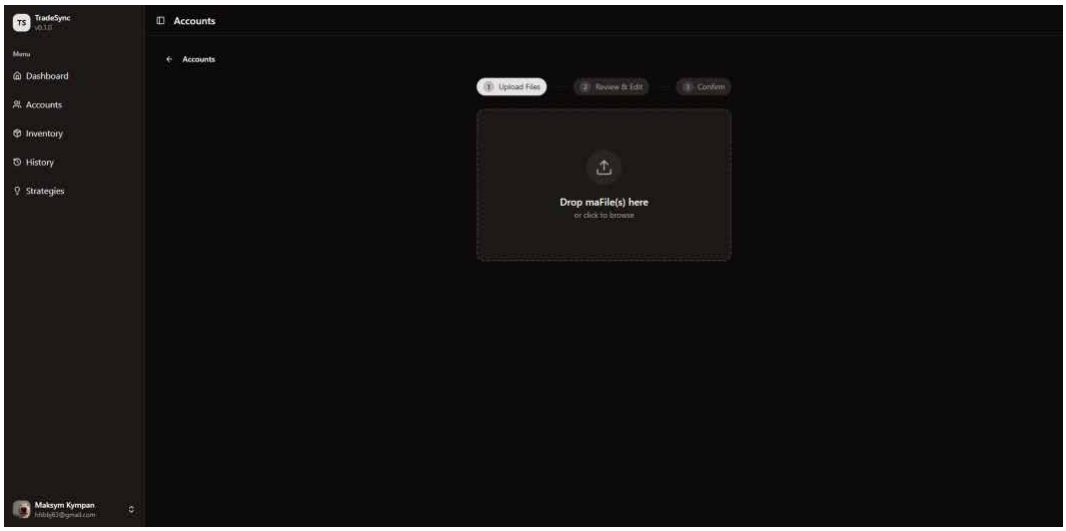

Рис. 24 Перший крок майстра імпорту – завантаження файлів .maFile

На другому кроці користувач вводить паролі до облікових записів у форматі «login:password» (рис. 25), на третьому – обирає тип кожного акаунту та підтверджує імпорт (рис. 26). Серверний обробник перевіряє Firebase ID-токен, виконує автентифікацію в Steam Network, шифрує конфіденційні поля алгоритмом AES-256-GCM та зберігає документи у Firestore.

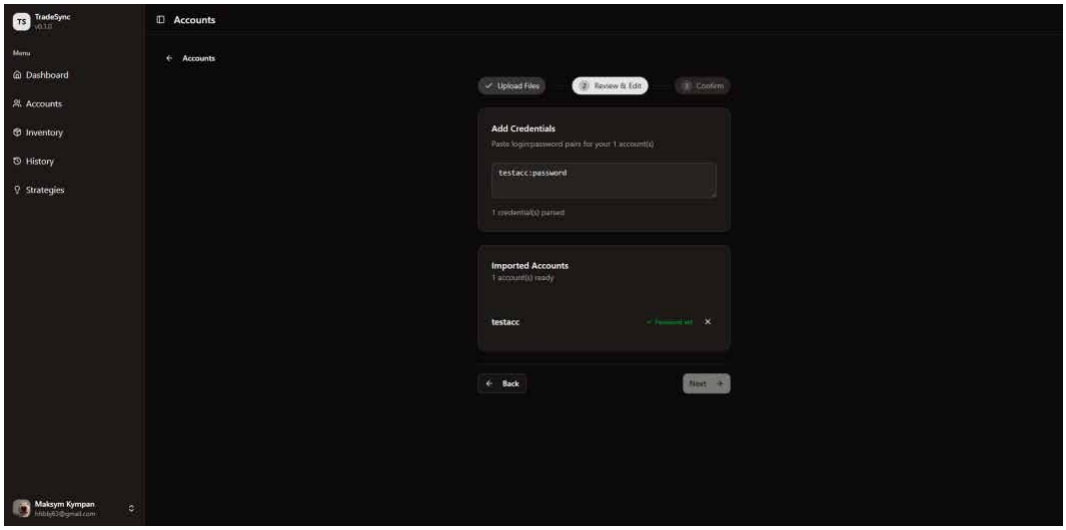

Рис. 25 Другий крок майстра імпорту – введення паролів облікових записів

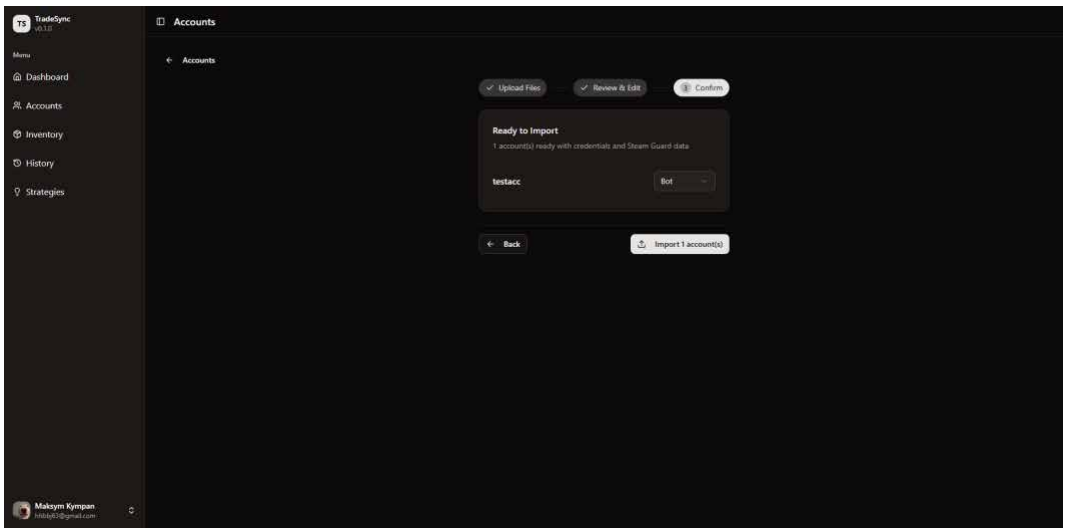

Рис. 26 Третій крок майстра імпорту – вибір типу облікового запису та підсумок

Після завершення майстра користувач повертається на сторінку керування обліковими записами, де нові записи відображаються разом із раніше імпортованими (рис. 27).

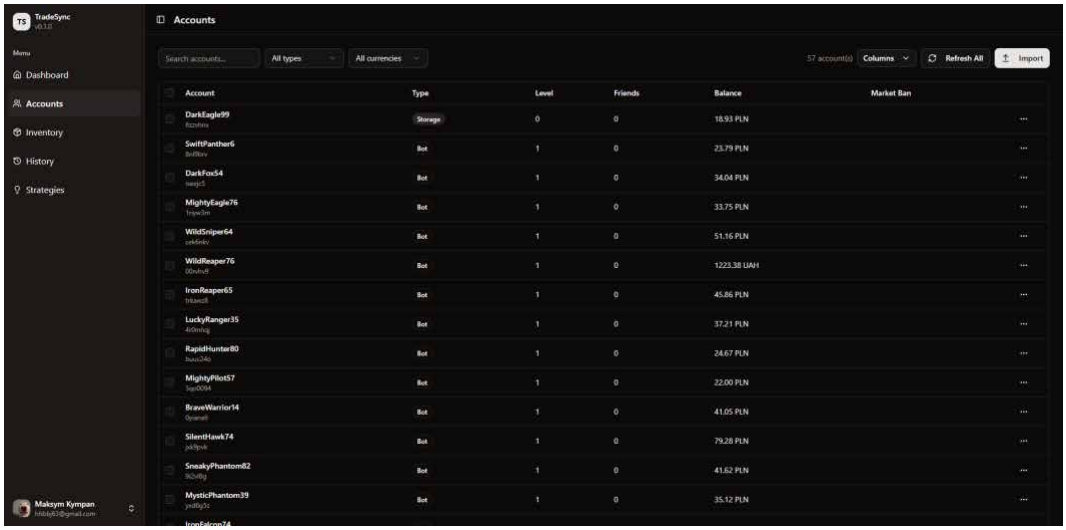

Рис. 27 Сторінка облікових записів з імпортованими даними

4. Перегляд інвентарю

У розділі «Inventory» відображаються віртуальні предмети, агреговані з усіх підключених облікових записів Steam, з характеристиками float, paint seed та цінами з підтримуваних торгових платформ (рис. 28).

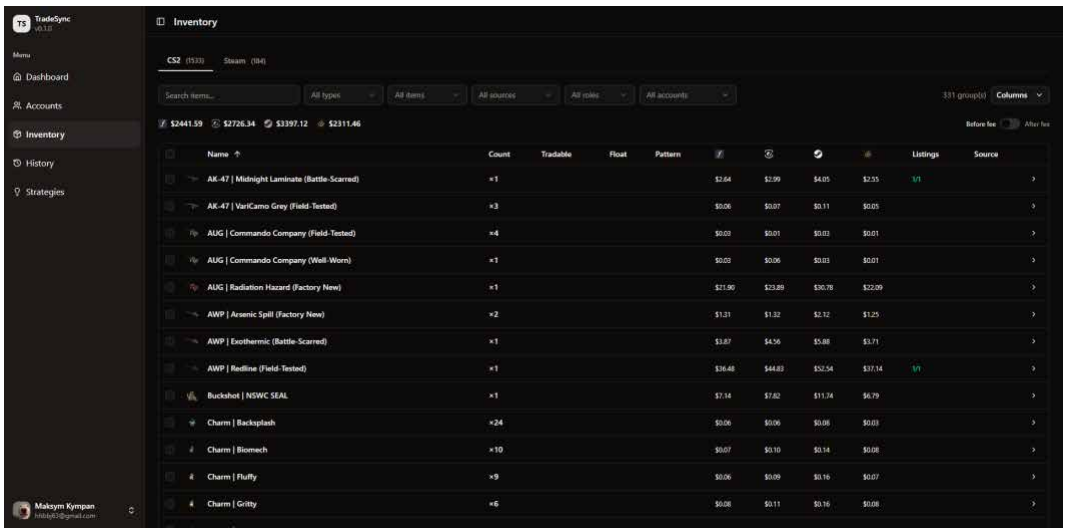

Рис. 28 Сторінка інвентарю з агрегованими даними з усіх облікових записів

5. Виставлення предмета на продаж

Користувач обирає предмет з інвентарю та натискає кнопку виставлення на продаж – відкривається діалогове вікно з вибором платформ і полями для введення ціни (рис. 29). Після підтвердження серверний обробник одночасно публікує предмет на обраних майданчиках через клієнти Steam Community, CSFloat, Market.CSGO та Youpin.

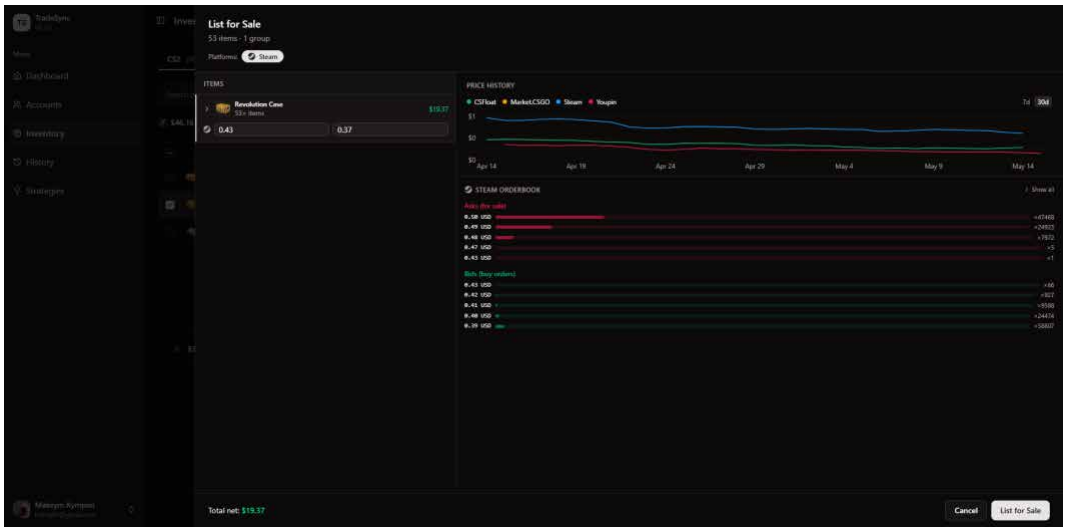

Рис. 29 Діалогове вікно виставлення предмета на продаж

6. Перегляд історії торгових операцій

У розділі «History» відображається повна таблиця подій інвентаря: KPI-плитки (Net P&L, Items sold, Avg margin, Purchases), тулбар фільтрів за назвою/періодом/типом події/платформою/акаунтом і пагінована таблиця подій

з колонками дати, предмета, типу, ціни, комісії, собівартості та чистого прибутку (рис. 30).

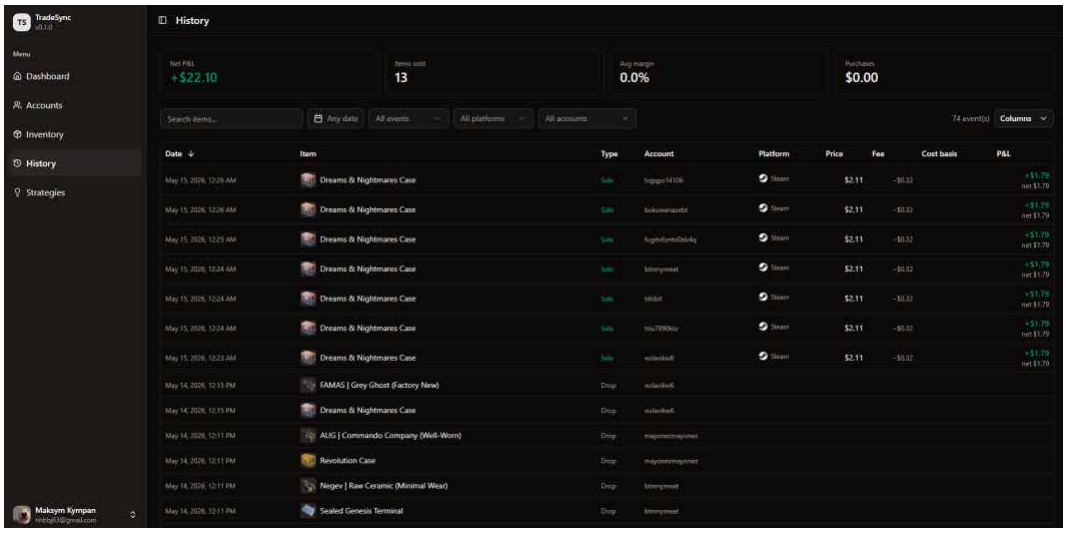

Рис. 30 Сторінка історії торгових операцій з фільтрами та таблицею подій

Результат тестування:

Система успішно пройшла всі етапи наведеного сценарію без збоїв. Авторизація через Firebase Authentication виконувалась коректно, майстер імпорту облікових записів послідовно проходив усі три кроки, конфіденційні поля шифрувались перед записом до Firestore, документи інвентаря синхронізувались між серверною та клієнтською частинами у режимі реального часу, виставлення предметів на торгові платформи завершувалось успішно, а історія торгових операцій формувалась з коректними значеннями прибутковості. Виявлені у процесі тестування дрібні зауваження до інтерфейсу та обробки мережевих помилок було усунено до завершення розробки.

4.2 Вимоги до апаратного та програмного забезпечення

Для коректного функціонування розробленої системи автоматизації торгівлі віртуальними предметами необхідно дотримуватися певних вимог до апаратного та програмного забезпечення, що забезпечить стабільність, швидкість роботи та безпеку даних. Правильна організація технічної бази дозволяє системі ефективно обробляти запити користувачів, підтримувати

одночасну роботу з декількома обліковими записами Steam, виконувати моніторинг ринкових пропозицій на кількох торгових платформах та реагувати на появу недооцінених лотів з мінімальною затримкою.

На рис. 31 наведено діаграму розгортання системи, на якій програмні компоненти розподілено між п'ятьма фізично відокремленими середовищами виконання: пристроями кінцевих користувачів, мережею Cloudflare, віддаленим віртуальним сервером, хмарною інфраструктурою Firebase та зовнішніми торговими сервісами. Така розподілена архітектура дозволяє масштабувати окремі компоненти незалежно один від одного.

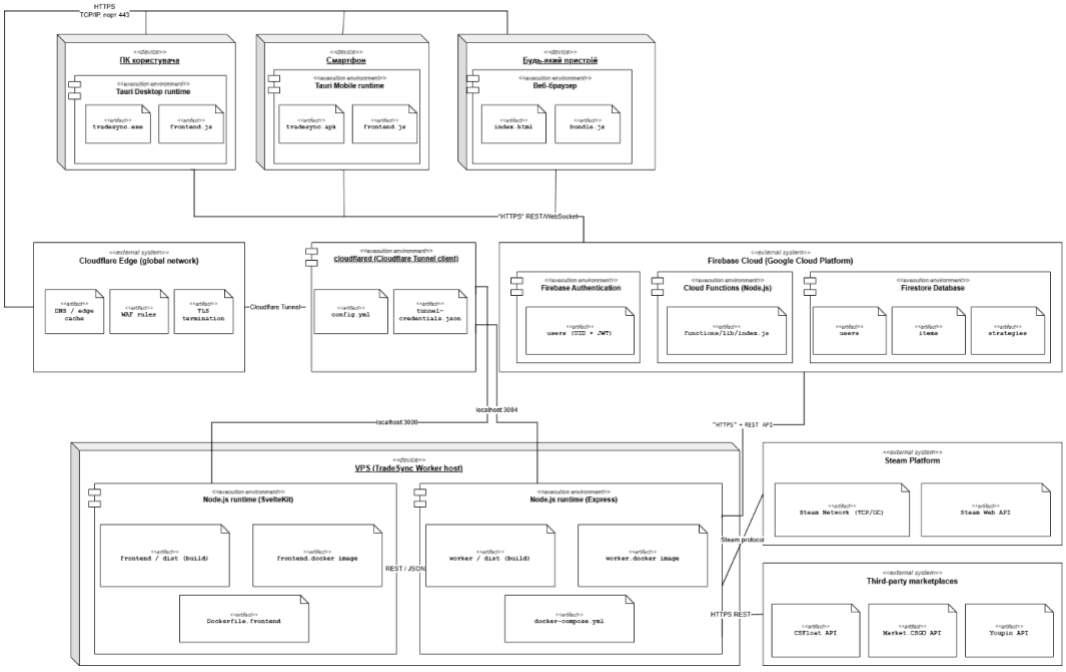

Рис. 31 Діаграма розгортання розробленої системи

Клієнтська частина виконується безпосередньо на пристроях користувачів у трьох форматах поставки: десктопний застосунок Tauri 2.0 для операційних систем Windows, macOS та Linux; мобільний застосунок Tauri 2.0 для Android та iOS; вебзастосунок, що завантажується у браузер за доменом системи. Усі три формати збираються з єдиної кодової бази SvelteKit 5.0 через адаптер adapter-static у вигляді односторінкового застосунку (SPA), що забезпечує однакову поведінку незалежно від обраного середовища.

Серверна частина розгорнута на віддаленому віртуальному сервері та складається з двох паралельних середовищ виконання Node.js: одне обслуговує веб-версію застосунку як статичний SPA-бандл SvelteKit, друге реалізує бізнес-логіку торгівлі через Express-обробник з модулями інтеграції зі Steam Network та сторонніми торговими платформами. Обидва середовища пакуються у Docker-контейнери та оркеструються через docker-compose, що забезпечує переносимість конфігурації та ізоляцію від хост-системи.

Зовнішній доступ до серверної частини здійснюється виключно через Cloudflare Tunnel – клієнт cloudflared встановлює зашифроване вихідне з'єднання з глобальною мережею Cloudflare, яка приймає HTTPS-запити на порту 443 та виконує TLS-термінацію, DNS-резолвінг і захист на рівні веб-фаєрволу. Така схема виключає необхідність відкривати публічні порти на сервері та керувати SSL-сертифікатами на боці сервера.

Хмарна інфраструктура Firebase у складі єдиного проєкту надає три сервіси: Authentication для автентифікації користувачів через Google OAuth з видачею JWT-токенів сесії, Firestore як основне документоорієнтоване сховище даних та Cloud Functions для виконання тригерних функцій у відповідь на події бази даних. Зовнішні торгові сервіси охоплюють Steam Network та Steam Web API для роботи з ігровою платформою, а також REST-інтерфейси CSFloat, Market.CSGO та Youpin для отримання ринкових даних зі сторонніх платформ.

Нижче наведені основні вимоги до апаратного та програмного забезпечення розробленої системи, згруповані за середовищами розгортання.

Вимоги до серверної частини

Серверна частина розгортається на віддаленому віртуальному сервері з підключенням до мережі Інтернет. Підтримка одночасної роботи з десятками активних Steam-сесій та постійного моніторингу ринкових пропозицій вимагає відповідного запасу обчислювальних ресурсів.

Апаратне забезпечення:

- процесор від 4 vCPU або аналогічний;
- оперативна пам'ять – 8 ГБ і більше;

- вільне місце на накопичувачі – не менше 40 ГБ SSD;
- мережеве підключення з пропускною здатністю від 100 Мбіт/с;
- доступ до мережі Інтернет без обмежень на вихідні з'єднання.

Програмне забезпечення:

- операційна система Ubuntu 22.04 / 24.04 LTS або сумісний дистрибутив Linux;
- Docker Engine версії 24.0 або новішої;
- Docker Compose для оркестрації контейнерів;
- середовище виконання Node.js версії 20 LTS;
- клієнт cloudflared для встановлення Cloudflare Tunnel.

Вимоги до клієнтської частини

Клієнтська частина встановлюється на пристроях кінцевих користувачів або відкривається у браузері. Вимоги до апаратного та програмного забезпечення залежать від обраного формату поставки.

Десктопний застосунок (Windows / macOS / Linux):

- 64-розрядний процесор Intel Core i3 / AMD Ryzen 3 або новіший;
- оперативна пам'ять – 4 ГБ і більше;
- вільне місце на накопичувачі – не менше 500 МБ;
- операційна система Windows 10 (або новіша), macOS 12 (або новіша) чи сучасний дистрибутив Linux;
- стабільне підключення до мережі Інтернет.

Мобільний застосунок (Android / iOS):

- Android 10 (або новіша версія) чи iOS 15 (або новіша);
- оперативна пам'ять – 2 ГБ і більше;
- вільне місце у пам'яті пристрою – не менше 200 МБ;
- стабільне підключення до мережі Інтернет.

Вебзастосунок (без встановлення):

- сучасний браузер: Google Chrome 100 (або новіший), Microsoft Edge 100 (або новіший), Mozilla Firefox 100 (або новіший), Safari 15 (або новіший);

- пристрій з підтримкою графічного інтерфейсу та доступом до мережі Інтернет;
- дійсний обліковий запис Google для входу до системи.

Вимоги до зовнішніх сервісів

Для повноцінної роботи розробленої системи необхідні підключення до зовнішніх сервісів, без яких функціонування основних модулів неможливе. Ці сервіси не входять до складу інсталяційного пакету і потребують окремої реєстрації.

- Обліковий запис Firebase з активованими сервісами Authentication, Firestore та Cloud Functions; для серверної частини необхідний JSON-ключ службового облікового запису Firebase Admin SDK;
- Обліковий запис Cloudflare із зареєстрованим доменом для системи та налаштованим Cloudflare Tunnel; токен тунелю записується у конфігурацію клієнта cloudflared;
- Облікові записи Steam з активованою двофакторною автентифікацією через Steam Mobile Authenticator та експортованими файлами .maFile, які користувач завантажує до системи через майстер імпорту;
- Програмні інтерфейси торгових платформ CSFloat, Market.CSGO та Youpin – для роботи з кожною з них необхідно отримати API-ключ через особистий кабінет на відповідному майданчику. Ключі вводяться користувачем у налаштуваннях кожного облікового запису.

Дотримання наведених вимог гарантує стабільну та продуктивну роботу всіх розподілених компонентів системи, зручність роботи кінцевих користувачів та надійний захист конфіденційних даних облікових записів Steam.

4.3 Склад інсталяційного пакету

Підготовлений інсталяційний пакет дозволяє за єдиним сценарієм розгорнути серверну інфраструктуру і встановити клієнтський застосунок на

пристроях кінцевих користувачів. Пакет містить усі необхідні компоненти для повноцінної роботи та постачається у двох незалежних наборах – окремо для серверної частини, призначеної для адміністратора системи, та окремо для кінцевих користувачів у вигляді готових бінарних пакетів для відповідних платформ.

Склад серверної частини пакета

До складу серверного пакета входять наступні компоненти.

- Вихідний код серверного обробника – каталог worker з реалізацією бізнес-логіки торгівлі, модулями інтеграції зі Steam Network та сторонніми торговими платформами, а також HTTP-маршрутами Express-додатка;
- Вихідний код клієнтського застосунку – каталог frontend з реалізацією клієнтської частини на SvelteKit 5.0, що збирається через адаптер adapter-static у вигляді SPA-бандлу та одночасно слугує джерелом для нативних збірок Tauri 2.0;
- Конфігурація контейнеризації – файли Dockerfile для обох runtime'ів та docker-compose.yml для оркестрації, що забезпечує автоматичне розгортання обох сервісів єдиною командою;
- Конфігурація Cloudflare Tunnel – файл config.yml з правилами маршрутизації трафіку та шаблон файлу tunnel-credentials.json, який заповнюється адміністратором;
- Конфігурація Firebase – файл firestore.rules з правилами безпеки доступу до бази даних, файл firestore.indexes.json з декларацією композитних індексів та каталог functions з вихідним кодом тригерних функцій Cloud Functions;
- Шаблон файлу .env – зразок налаштувань середовища виконання з переліком необхідних змінних: ключ шифрування облікових даних, шлях до JSON-ключа Firebase Admin SDK, ідентифікатор Cloudflare Tunnel та параметри інтеграції з торговими платформами;

- Документація – файл README.md з інструкцією з розгортання серверної частини, переліком потрібних зовнішніх облікових записів та послідовністю кроків ініціалізації системи.

Склад клієнтської частини пакета

Клієнтський застосунок постачається у нативних форматах для кожної підтримуваної платформи: інсталятори для Windows (.msi, .exe), образи для macOS (.dmg), пакети для Linux (.deb, .rpm, .AppImage), Android (.apk, .aab) та iOS (.ipa). Усі бінарні файли формуються вбудованим інструментарієм Tauri 2.0 з єдиної кодової бази. Окремо доступна вебверсія, розгорнута на серверній частині, що не потребує встановлення.

Особливості розгортання

- розгортання серверної частини виконується одноразово адміністратором системи через команду docker-compose up, яка автоматично збирає та запускає обидва Node.js-середовища у Docker-контейнерах;
- перед першим запуском серверної частини необхідно зареєструвати проєкт у Firebase Console з активацією сервісів Authentication, Firestore та Cloud Functions, а також згенерувати JSON-ключ службового облікового запису Firebase Admin SDK;
- тригерні функції Cloud Functions розгортаються окремою командою Firebase CLI, після чого автоматично виконуються у відповідь на події бази даних без додаткової інфраструктури;
- клієнтська частина у форматі бінарних пакетів встановлюється кінцевим користувачем стандартними засобами операційної системи без необхідності додаткових налаштувань;
- для адміністрування бази даних рекомендується використовувати консоль Firebase Console, яка надає графічний інтерфейс перегляду документів колекцій users, accounts, items, events та strategies.

Таким чином, інсталяційний пакет забезпечує комплексне розгортання та налаштування як серверної, так і клієнтської частин розробленої системи, що

гарантує їхню коректну взаємодію та готовність до експлуатації в робочому середовищі.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення для автоматизації торгівлі віртуальними предметами» пройдено всі ключові етапи створення програмного продукту – від дослідження предметної області та проєктування інформаційного забезпечення до програмної реалізації, тестування й підготовки інсталяційних артефактів.

Проведений аналіз ринку віртуальних предметів платформи Steam та сторонніх торгових майданчиків дозволив визначити ключові напрями автоматизації, які включають одночасну роботу з кількома обліковими записами Steam, моніторинг ринкових цін на кількох торгових платформах, автоматичне виявлення недооцінених лотів за параметрами зношеності та шаблону, виконання пропозицій обміну, а також облік торгових операцій з розрахунком чистого прибутку. Огляд аналогічних систем (Skinledger, TradeOn Ultra, SteamLedger) показав, що жодна з них не забезпечує повного циклу автоматизованої торгівлі у поєднанні з координованим керуванням кількома акаунтами та спеціалізованими алгоритмами пошуку лотів за характеристиками предметів, що обґрунтувало доцільність розробки нового програмного забезпечення. На основі виявлених потреб сформовано постановку завдання та розроблено архітектурну концепцію системи, яка лягла в основу подальшої реалізації.

Було побудовано логічну модель даних у вигляді ER-діаграми з п'яти ключових сутностей (користувачі, Steam-акаунти, віртуальні предмети, події інвентаря та правила автоматичної купівлі), спроєктовано фізичну структуру бази у документоорієнтованому сховищі Google Cloud Firestore, а також задекларовано композитні індекси, тригерні функції та правила безпеки доступу. Застосування Firestore у поєднанні з Firebase Authentication та Cloud Functions дозволило винести розмежування доступу та підтримку референційної цілісності на рівень самого сховища, що зменшило загальну кількість зовнішніх залежностей системи та виключило цілий клас

вразливостей, пов'язаних з помилками в логіці контролю доступу на стороні клієнта.

Клієнтська частина програмного забезпечення реалізована мовою TypeScript у середовищі Visual Studio Code з використанням фреймворку SvelteKit 5.0 та бібліотеки компонентів Shadcn-svelte зі стилізацією через TailwindCSS. Для пакування застосунку у нативні виконувані файли застосовано платформу Tauri 2.0, що дозволило з єдиної кодової бази сформувати десктопний застосунок для Windows, macOS та Linux, мобільний застосунок для Android та iOS, а також вебверсію, доступну через браузер без додаткового встановлення. Такий вибір забезпечив розробку сучасного, зручного інтерфейсу з мінімальним споживанням системних ресурсів та однаковою поведінкою на всіх підтримуваних платформах.

Серверна частина реалізована на платформі Node.js з використанням фреймворку Express та структурно поділена на чотири модулі: модуль роботи зі Steam Network на основі бібліотек steam-user, steam-totp та steamcommunity, модуль торгових операцій на основі steam-tradeoffer-manager і globaloffensive, модуль інтеграції зі сторонніми торговими платформами CSFloat, Market.CSGO та Youpin, а також модуль ціноутворення та валютної конверсії. Зовнішній доступ до серверної частини забезпечується через Cloudflare Tunnel без відкриття публічних портів на сервері, а конфіденційні дані облікових записів Steam зберігаються виключно у зашифрованому вигляді з використанням алгоритму AES-256-GCM.

Центральною функціональною складовою розробленої системи є модуль автоматичного виявлення недооцінених пропозицій на ринку. На основі активних правил купівлі, які користувач налаштовує через графічний інтерфейс, система у режимі реального часу аналізує ринкові пропозиції з кількох торгових платформ та автоматично здійснює купівлю лотів, що відповідають заданим діапазонам зношеності, шаблону та максимальної ціни. Такий підхід дозволяє суттєво підвищити ефективність торгової діяльності учасника ринку та зменшити вплив людського фактору на її результати.

Розроблене програмне забезпечення повністю готове до експлуатації: до нього додається інсталяційний пакет, який забезпечує оперативне розгортання серверної та клієнтської частин. Серверна частина розгортається у Docker-контейнерах єдиною командою docker-compose, клієнтська частина постачається у вигляді нативних збірок для кожної з підтримуваних платформ, а вебверсія доступна без додаткового встановлення. Запуск системи виконується кількома командами та не потребує спеціалізованої експертизи, що зменшує тривалість підготовчого етапу впровадження.

Сформульована у вступі мета досягнута у повному обсязі, а перелічені дослідницькі та інженерні завдання – реалізовані. Розроблена програмна система успішно автоматизує ключові процеси учасника ринку віртуальних предметів, поєднуючи кросплатформність клієнтської частини, стабільність роботи розподіленої архітектури, надійне шифрування конфіденційних даних та можливості автоматичного виявлення недооцінених пропозицій на ринку. Реалізована система є готовим до експлуатації продуктом, що демонструє практичну ефективність на ринку віртуальних предметів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pricempire. CS2 Market Insights and Statistics. URL: https://pricempire.com/ (дата звернення: 15.05.2026).

2. Steam Inventory Service. Steamworks Documentation. Valve Corporation. URL: https://partner.steamgames.com/doc/features/inventory (дата звернення: 15.05.2026).

3. Дудзяний І. М. Об'єктно-орієнтоване моделювання програмних систем: навчальний посібник. Львів: Видавничий центр ЛНУ імені Івана Франка, 2007. 108 с.

4. OMG. OMG® Unified Modeling Language® (OMG UML) Specification, Version 2.5.1. Object Management Group, 2017. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: 15.05.2026).

5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston: Addison-Wesley Professional, 2003. 208 p.

6. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3rd ed. Upper Saddle River: Prentice Hall, 2004. 736 p.

7. Skinledger – Multi-Account Steam Inventory Manager. URL: https://skinledger.com/ (дата звернення: 15.05.2026).

8. TradeOn Ultra Documentation. URL: https://tradeon-guide.gitbook.io/tradeon-ultra (дата звернення: 15.05.2026).

9. SteamLedger – Portfolio & Arbitrage Tracker. URL: https://steamledger.com/ (дата звернення: 15.05.2026).

10. Чен П. П. Модель «сутність-зв'язок» – крок до єдиного бачення даних. ACM Transactions on Database Systems. 1976. Т. 1, № 1. С. 9–36.

11. Sadalage P. J., Fowler M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Boston: Addison-Wesley Professional, 2013. 192 p.

12. Google. Cloud Firestore Documentation. URL: https://firebase.google.com/docs/firestore (дата звернення: 15.05.2026).

13. Zod – TypeScript-first schema validation. URL: https://zod.dev/ (дата звернення: 15.05.2026).

14. Google. Cloud Firestore Index Configuration. URL: https://firebase.google.com/docs/firestore/query-data/indexing (дата звернення: 15.05.2026).

15. Google. Cloud Functions for Firebase: Firestore triggers. URL: https://firebase.google.com/docs/functions/firestore-events (дата звернення: 15.05.2026).

16. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston: Addison-Wesley Professional, 2005. 475 p.

17. Tanenbaum A. S., Van Steen M. Distributed Systems: Principles and Paradigms. 2nd ed. Upper Saddle River: Pearson Prentice Hall, 2007. 686 p.

18. Dworkin M. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D. Gaithersburg: National Institute of Standards and Technology, 2007. 39 p. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf (дата звернення: 15.05.2026).

19. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2015. 816 p.

20. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 8th ed. New York: McGraw-Hill, 2014. 976 p.

21. Beizer B. Black-Box Testing: Techniques for Functional Testing of Software and Systems. New York: John Wiley & Sons, 1995. 320 p.

ДОДАТОК А

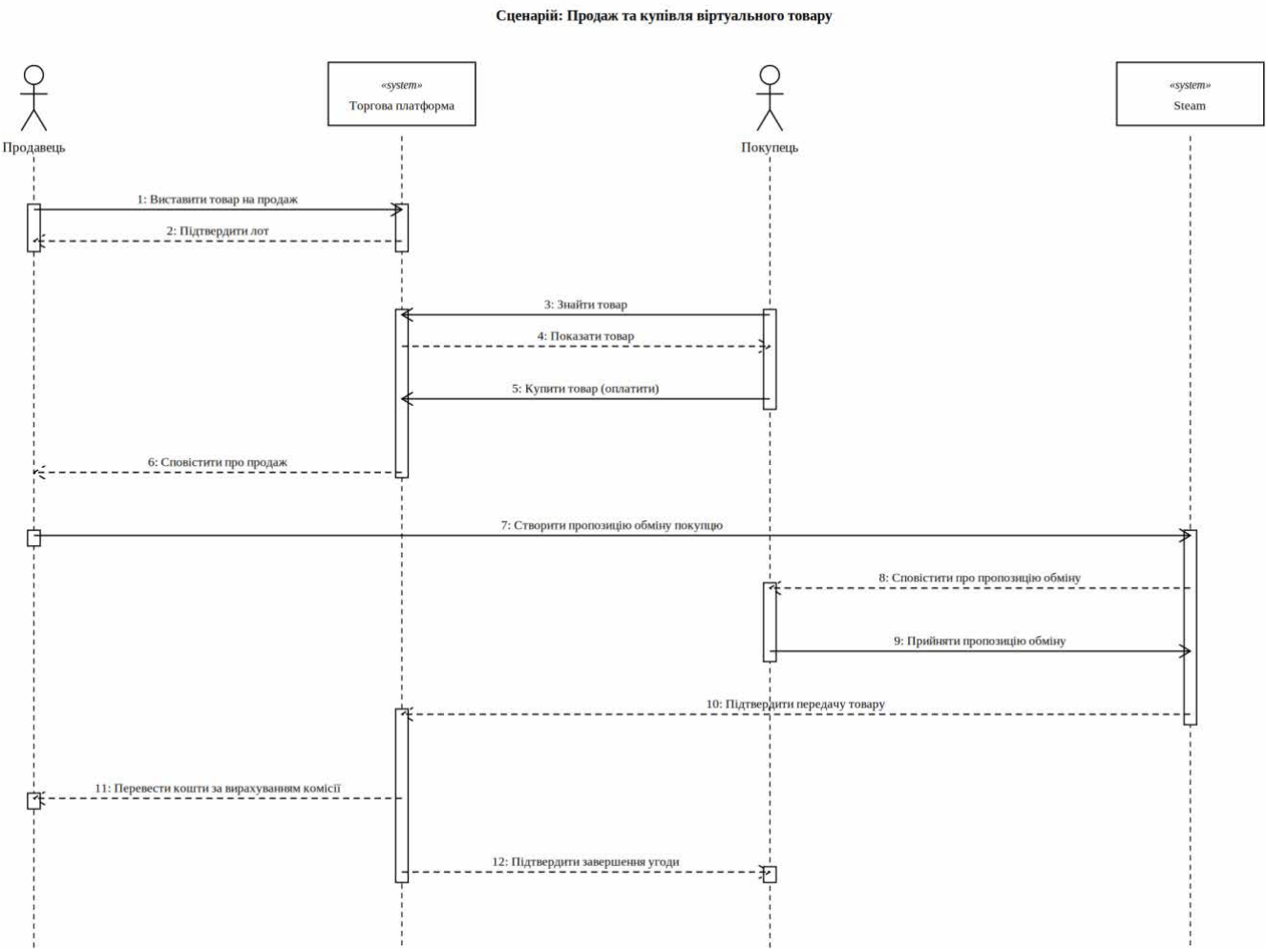

Рис. А.1 Діаграма послідовності продажу та купівлі віртуального товару

ДОДАТОК Б

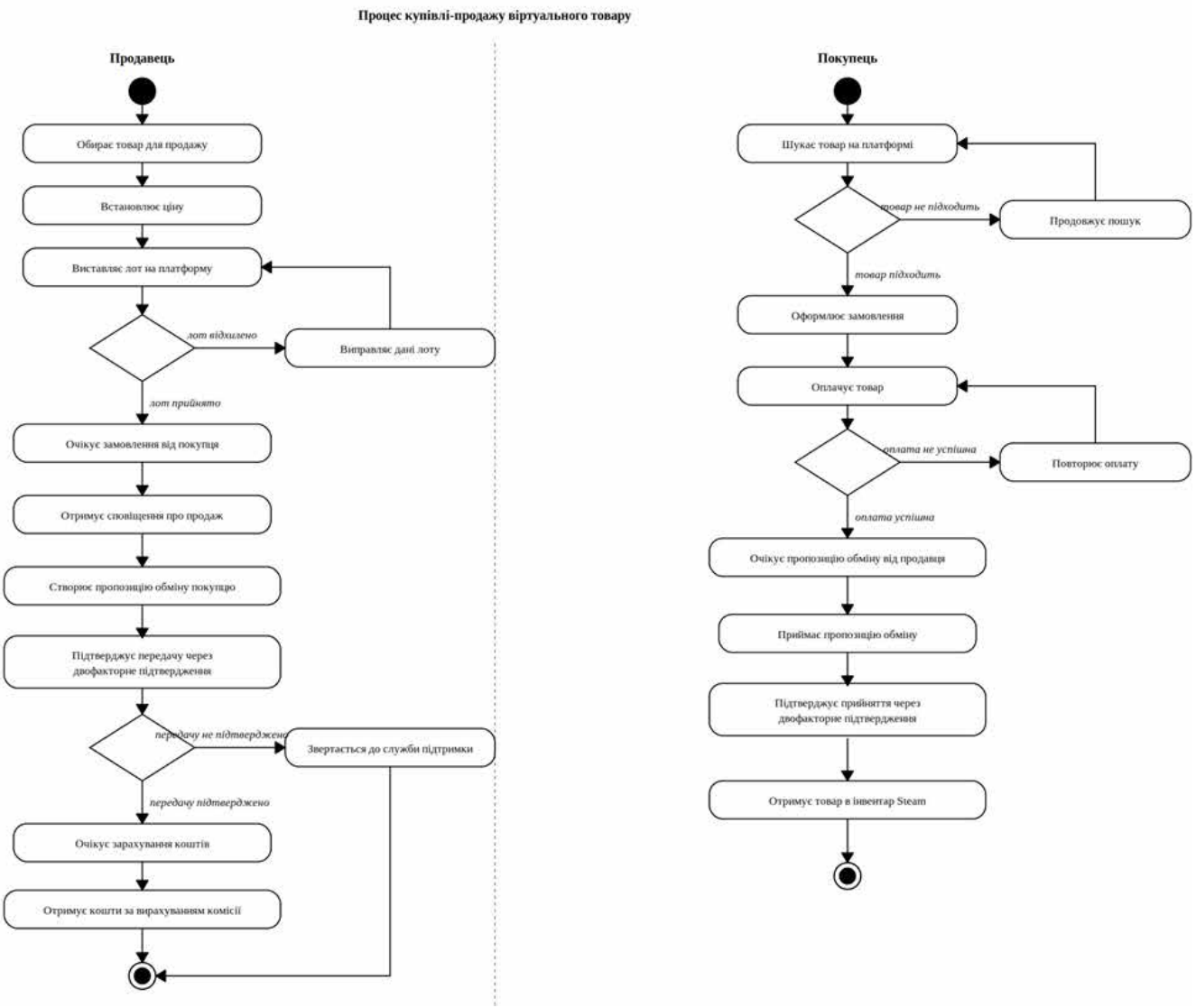

Рис. Б.1 Діаграма активності процесу купівлі-продажу віртуального товару

ДОДАТОК В

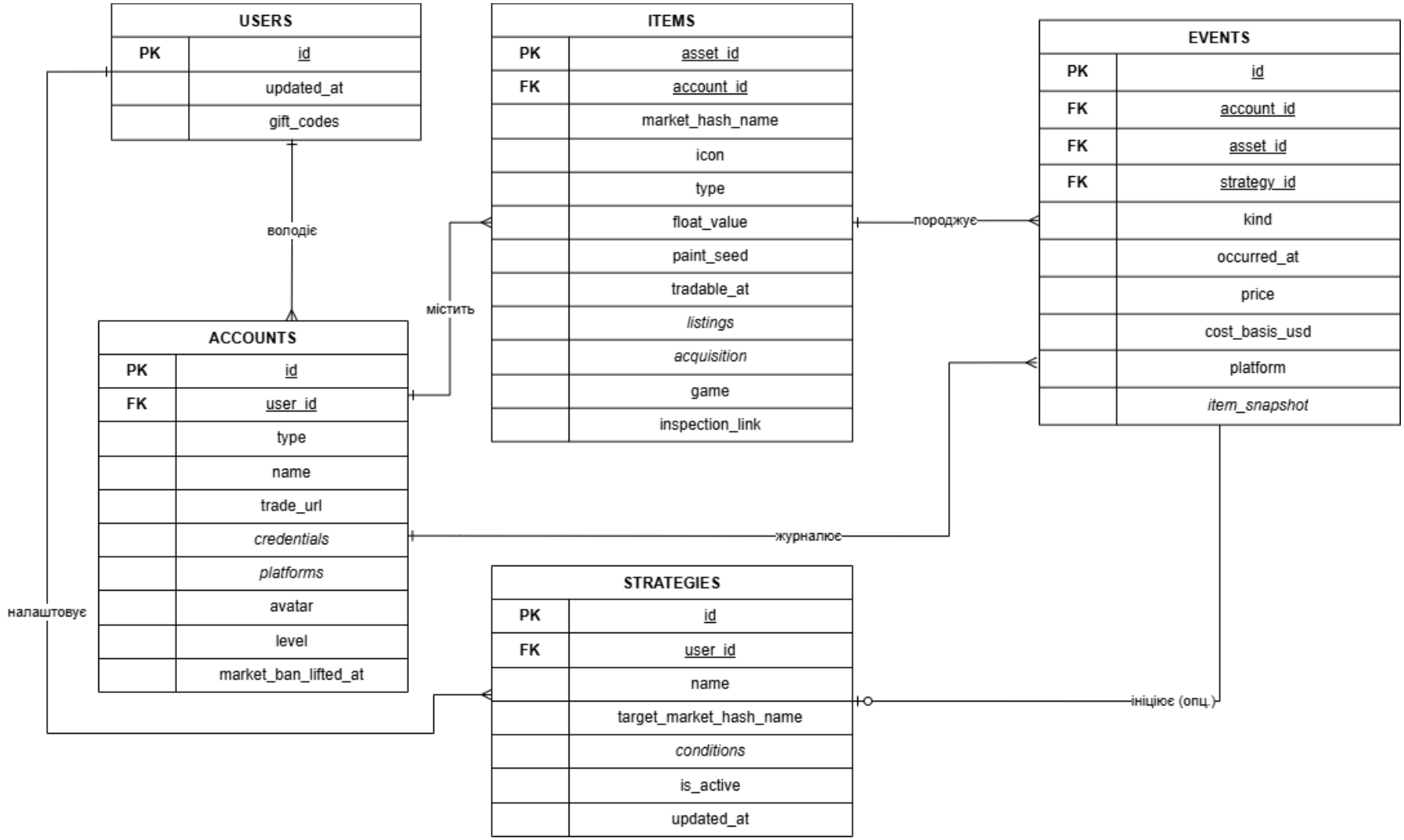

Рис. В.1 ER-діаграма інформаційної бази системи

ДОДАТОК Д

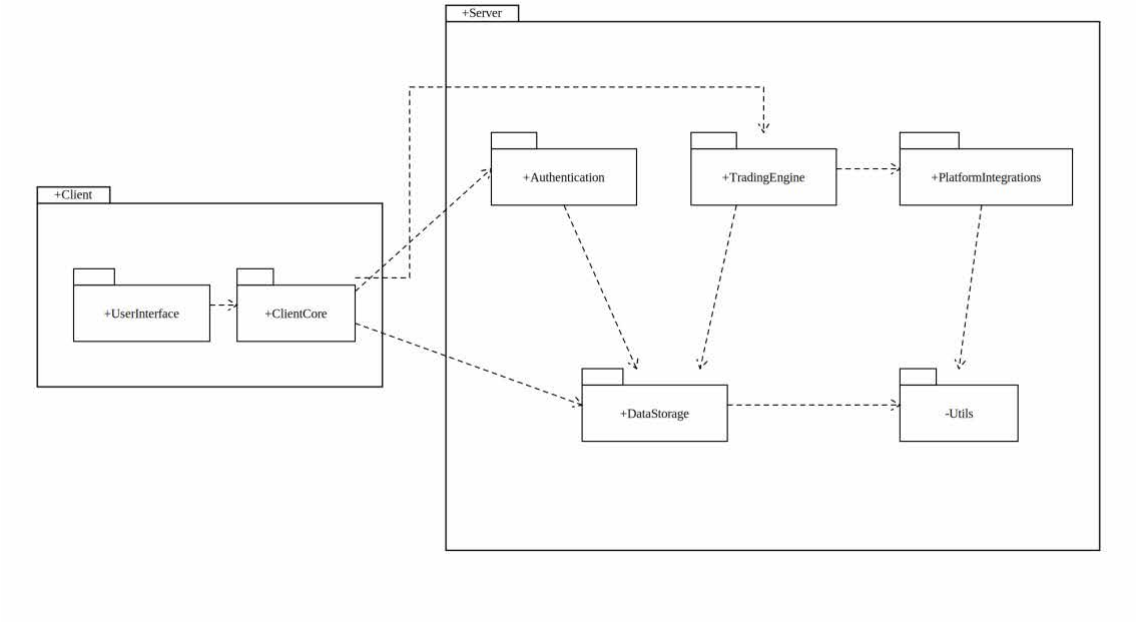

Рис. Д.1 Діаграма пакетів програмного забезпечення

ДОДАТОК Е

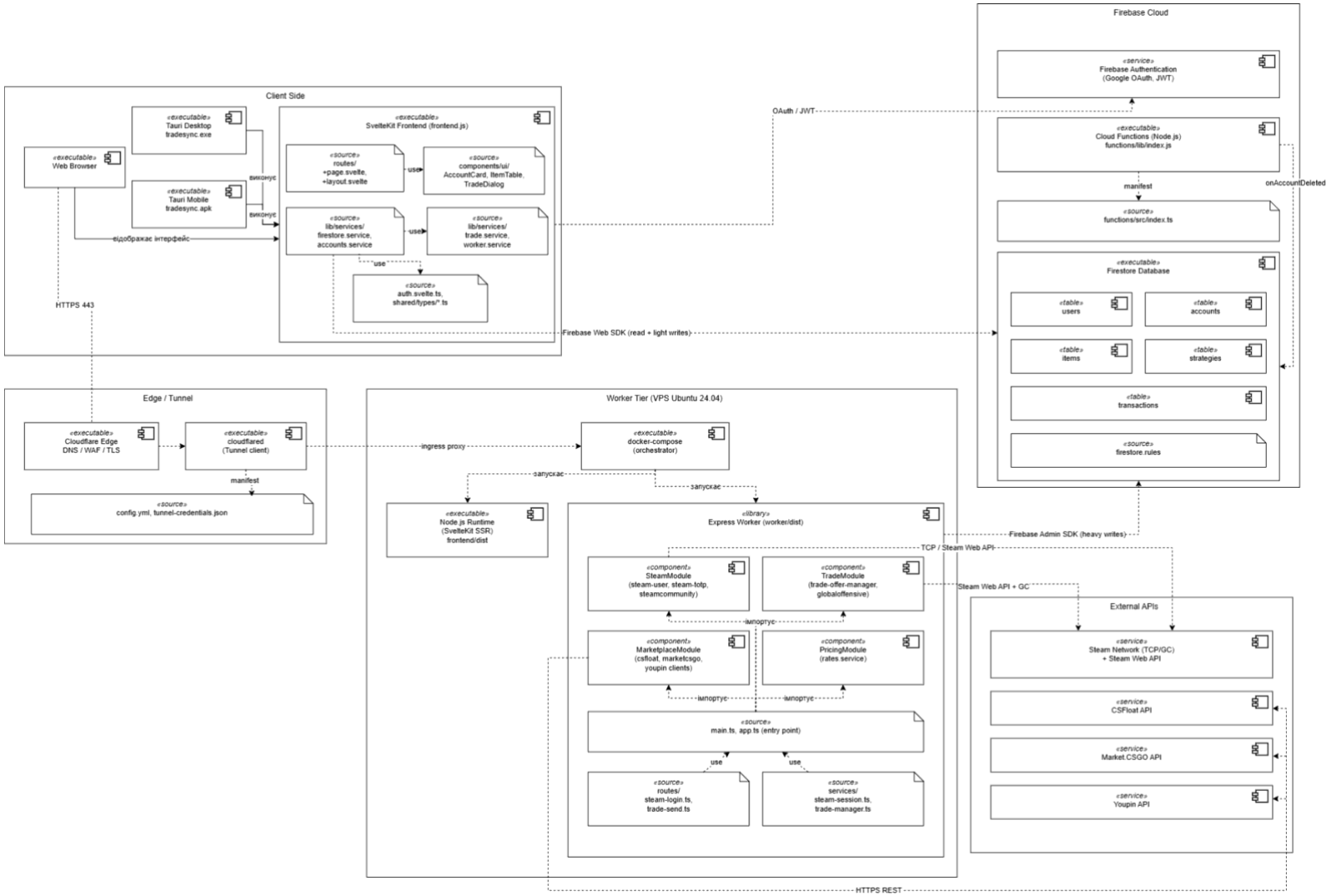

Рис. Е.1 Діаграма компонентів програмного забезпечення

ДОДАТОК Ж

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Кумпан Максим Вадимович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення для автоматизації торгівлі віртуальними предметами» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [×] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [×] перекладу іншомовних джерел;
  - [×] стилістичного редагування та перевірки граматики;
  - [×] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___»_____ 2026 р. _____ **Максим КУМПАН**