МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ ДЛЯ ФІНАНСОВОГО МОНІТОРИНГУ КОЛЕКЦІЙНИХ ЦИФРОВИХ ОБ'ЄКТІВ НА БАЗІ ANGULAR ТА ASP.NET CORE»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
О.О. Ткаченко
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент Директор ТОВ Голден-Тім Калін М.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ткаченку Олегу Олексійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційно-аналітичної системи для фінансового моніторингу колекційних цифрових об’єктів на базі Angular та ASP.NET Core

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та існуючих рішень.
2. Проєктування системи.
3. Реалізація та тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	28.01.2026	
3.	Обробка методичних та теоретичних	20.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	08.04.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2026	

Студент _____ О.О. Ткаченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційно-аналітичної системи для фінансового моніторингу колекційних цифрових об’єктів на базі Angular та ASP.NET Core»: 46 с., 24 рис., 15 джерел, 1 додаток.

АНАЛІТИКА ЦИФРОВИХ АКТИВІВ, АНАЛІЗ ФІНАНСОВИЙ, АВТОМАТИЗОВАНИЙ ЗБІР ДАНИХ, ГРАФІКИ ВАРТОСТІ, ІНВЕСТИЦІЙНИЙ ПОРТФЕЛЬ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, РИНКОВІ ДАНІ, ЦИФРОВІ АКТИВИ, ANGULAR, COUNTER-STRIKE 2, PNL, REST API, ROI, STEAM OPENID.

Об’єкт дослідження – процес розроблення та супроводу програмного забезпечення для аналізу цифрових активів.

Мета роботи: розроблення веборієнтованої системи для аналізу інвестиційних портфелів цифрових активів гри Counter-Strike 2, яка забезпечує автоматизований збір ринкових даних, розрахунок фінансових показників (PnL, ROI) та візуалізацію зміни вартості активів для підвищення ефективності оцінювання інвестицій користувачів.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

В результаті отримано сервіс для відстеження та аналізу цифрових активів засобами Angular Та ASP.NET Core. Сервіс повністю забезпечує автоматизований збір ринкових даних, розрахунок фінансових показників та відображення результатів аналізу в зручному вигляд. Отримані результати можуть бути використанні для створення та впровадження повноцінного вебсервісу.

SUMMARY

Bachelor’s qualifying paper “Development of an Information and Analytical System for Financial Monitoring of Collectible Digital Assets based on Angular and ASP.NET Core”: 46 pages, 24 figures, 15 references, 1 supplement.

ANGULAR, AUTOMATED DATA COLLECTION, CLIENT-SERVER ARCHITECTURE, COUNTER-STRIKE 2, DIGITAL ASSETS, FINANCIAL ANALYSIS, INVESTMENT PORTFOLIO, MARKET DATA, PNL, REST API, ROI, STEAM OPENID, VALUE CHARTS.

The object of the study is the process of developing and maintaining software for analyzing digital assets.

The aim of the study is development of a web-based system for analyzing investment portfolios of digital assets of the game Counter-Strike 2, which provides automated collection of market data, calculation of financial indicators (PnL, ROI), and visualization of changes in asset values to increase the efficiency of evaluating user investments.

The method of research includes methods of collecting and analyzing software requirements, methods of modelling, designing, constructing, and testing software.

The result is a service for tracking and analyzing digital assets using Angular Ta ASP.NET Core. The service fully provides automated collection of market data, calculation of financial indicators and display of analysis results in a convenient form. The results obtained can be used to create and implement a full-fledged web service.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Аналіз предметної області та існуючих рішень..... 10
1.1 Поняття цифрових активів та інвестиційного портфеля 10
1.2 Особливості ринку цифрових предметів Counter-Strike 2 11
1.3 Аналіз існуючих рішень 11
1.4 Постановка задачі 12
2 Проєктування системи..... 14
2.1 Аналіз та вибір архітектури 14
2.2 Діаграми системи 14
2.2.1 Діаграма варіантів використання..... 14
2.2.2 Діаграма класів..... 16
2.2.3 Діаграма компонентів..... 17
2.2.4 Діаграми послідовностей 17
2.3 Проєктування бази даних..... 19
3 Реалізація та тестування вебзастосунку..... 22
3.1 Реалізація авторизації та збору даних..... 22
3.1.1 Авторизація через Steam OpenID 22
3.1.2 Реалізація системи збору даних про предмети 23
3.2 Реалізація аналітики інвестиційного портфеля..... 26
3.2.1 Розрахунок PnL (Profit and Loss)..... 26
3.2.2 Розрахунок ROI (Return on Investment) 27
3.2.3 Візуалізація аналітичних даних (графіки портфеля) 28
3.3 Реалізація користувацького інтерфейсу 29
3.3.1 Головна сторінка..... 29

3.3.2 Сторінка перегляду предмета 30
3.3.3 Авторизація користувача через Steam 31
3.3.4 Сторінка портфелів користувача 32
3.3.5 Сторінка детального перегляду портфеля 33
3.4 Тестування серверної частини та опис API 35
3.4.1 Авторизація користувача 35
3.4.2 Управління портфелями 35
3.4.3 Аналітика портфеля 36
3.4.4 Робота з предметами 36
3.4.5 Додаткові сервіси системи 37
3.4.6 Безпека API та захист від типових вебвразливостей 37
3.4.7 Результати тестування 38
Висновки 40
Перелік посилань 41
Додаток А Приклади API-запитів та структур даних системи 43
А.1 Приклад отримання списку ігрових предметів 43
А.2 Приклад отримання даних інвестиційного портфеля 44
А.3 Приклад даних для побудови графіка динаміки інвестиційного портфеля 45

ВСТУП

У сучасних умовах розвитку інформаційних технологій та цифровізації економіки значної популярності набувають інвестиції у цифрові активи, зокрема предмети у багатокористувацьких онлайн-іграх. Однією з найпопулярніших платформ, яка надає можливості купувати, продавати та обмінювати такі предмети, є Steam, а однією з найпопулярніших багатокористувацьких ігор є Counter-Strike 2. Система рідкості, якості та ступеня зносу предметів створює стабільний попит серед гравців, а їх візуальна привабливість додатково підкріплює зацікавленість серед спільноти, що забезпечує динамічну зміну ринкової вартості предметів. На даний момент капіталізацію ринку предметів Counter-Strike 2 оцінюють у понад 6 мільярдів доларів США, а протягом останніх років ринок демонструє стійку тенденцію до зростання. Саме це створює необхідність у використанні системи для моніторингу, аналізу та оцінювання користувацьких портфелів.

Актуальність теми полягає у зростанні попиту на системи аналізу цифрових активів, які дозволяють користувачам отримувати інформацію про зміну цін, прибутковість предметів та загальний стан портфеля в одному місці. Існуючі сервіси мають обмежений функціонал, недостатню гнучкість або не забезпечують достатньо зручної візуалізації даних. Тому розроблення сучасної, веборієнтованої та зручної системи для пошуку та відстежування предметів та портфелів є актуальним завданням у сфері програмної інженерії.

Об’єктом дослідження є процес розроблення та супроводу програмного забезпечення для аналізу цифрових активів.

Предметом дослідження є методи та програмні засоби обробки, збереження, аналізу й візуалізації даних цифрових активів із використанням клієнт-серверної архітектури.

Метою роботи розроблення веборієнтованої системи для аналізу інвестиційних портфелів цифрових активів гри Counter-Strike 2, яка забезпечує

автоматизований збір ринкових даних, розрахунок фінансових показників (PnL, ROI) та візуалізацію зміни вартості активів для підвищення ефективності оцінювання інвестицій користувачів.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

– проаналізувати існуючі програмні рішення;
– визначити функціональні та нефункціональні вимоги до системи;
– спроєктувати архітектуру програмного забезпечення;
– реалізувати серверну частину системи для обробки та збереження даних;
– розробити методи аналізу прибутковості та зміни вартості предметів та портфелів користувача;
– провести тестування програмного забезпечення та оцінити результати його роботи.

Практичне значення отриманих результатів полягає у можливостівикористання розробленої системи для автоматизованого аналізу цифрових активів користувачів, відстеження змін ринкової вартості предметів та спрощення процесу оцінювання прибутковості портфеля.

У процесі розроблення програмного забезпечення були використані сучасні технології веброзробки, зокрема Angular для створення клієнтської частини, ASP.NET Core для реалізації серверної логіки, а також система керування базами даних Microsoft SQL Server для збереження інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття цифрових активів та інвестиційного портфеля

Цифрові активи – це нематеріальні об’єкти, які мають певну ринкову вартість у цифровому середовищі. У контексті ігрової індустрії до таких активів належать внутрішньоігрові предмети, наприклад скіни у грі Counter-Strike 2. Дані предмети мають унікальні характеристики, такі як рідкість, якість, ступінь зносу та зовнішній вигляд, що безпосередньо впливає на їхню ринкову ціну.

Ринок цифрових ігрових предметів функціонує за принципами класичного ринку активів: попит і пропозиція визначають вартість предметів, а їх ціна може змінюватися з часом. Це робить такі предмети потенційним об’єктом інвестування.

Інвестиційний портфель – це сукупність активів, які належать одному користувачу або інвестору та управляються як єдине ціле. У контексті даної роботи інвестиційний портфель складається з набору цифрових предметів (скінів), кожен з яких має свою кількість, ціну придбання та поточну ринкову вартість.

Основна мета інвестиційного портфеля полягає у відстеженні змін вартості активів та оцінці їх прибутковості. Для цього використовуються такі показники, як:

– PnL (Profit and Loss) – показник прибутку або збитку від інвестицій [11];
– ROI (Return on Investment) – відсоткова рентабельність вкладень [12].

Таким чином, інвестиційний портфель дозволяє користувачу систематизувати свої активи та аналізувати ефективність інвестицій.

1.2 Особливості ринку цифрових предметів Counter-Strike 2

Ринок предметів Counter-Strike 2 є одним із найбільших сегментів цифрової торгівлі у сфері відеоігор. Його особливістю є висока ліквідність та постійна зміна цін.

На вартість предметів впливають такі фактори:

– рідкість предмета;
– популярність серед гравців;
– зовнішній вигляд та якість;
– оновлення гри;
– загальний попит на ринку.

Через ці фактори ціни можуть змінюватися як у короткостроковій, так і у довгостроковій перспективі, що робить ринок складним для аналізу без спеціалізованих інструментів.

1.3 Аналіз існуючих рішень

На сьогодні існує кілька платформ, що дозволяють користувачам переглядати та аналізувати ціни на ігрові предмети.

Steam Community Market – офіційна торговельна платформа Steam, яка дозволяє купувати та продавати предмети. Вона надає базову інформацію про ціни та історію продажів, однак не містить інструментів для глибокої аналітики.

Недоліки:

– відсутність інвестиційних портфелів;
– немає розрахунку прибутковості;
– обмежена аналітика.

Сторонні аналітичні сервіси (Buff, CSFloat тощо) надають розширену інформацію про ринок, включаючи графіки цін та статистику.

Недоліки:

– відсутність персоналізованих портфелів;
– відсутність PnL та ROI для користувача;
– обмежені можливості управління активами.

Таким чином, існуючі рішення не забезпечують повного циклу роботи з інвестиційним портфелем користувача.

1.4 Постановка задачі

У межах даної роботи ставиться задача розроблення веборієнтованої інформаційної системи для аналізу інвестиційних портфелів цифрових активів у середовищі гри Counter-Strike 2.

Система повинна забезпечувати повний цикл роботи користувача з цифровими активами: від отримання актуальної інформації про предмети до аналізу ефективності інвестицій у вигляді аналітичних показників та графічної візуалізації даних.

Для досягнення поставленої мети необхідно реалізувати наступний функціонал:

– забезпечити механізм автентифікації користувачів через сервіс Steam OpenID з автоматичним створенням профілю користувача в системі;
– реалізувати модуль отримання та збереження інформації про цифрові предмети Counter-Strike 2 із зовнішніх джерел (Steam Market API);
– розробити функціонал перегляду каталогу предметів із можливістю пошуку та фільтрації;
– забезпечити створення, редагування та видалення інвестиційних портфелів користувачів;
– реалізувати механізм додавання та видалення предметів у межах конкретного портфеля;
– розробити алгоритми розрахунку фінансових показників інвестицій: прибуток або збиток (PnL), рентабельність інвестицій (ROI);

– забезпечити отримання історичних даних зміни вартості предметів для побудови графіків;
– реалізувати модуль аналітичної візуалізації даних у вигляді графіків зміни вартості портфеля у часі;
– забезпечити взаємодію клієнтської та серверної частини системи через REST API із використанням формату JSON;
– реалізувати систему захисту даних користувача та обмеження доступу до персональних інвестиційних даних.

Додатково система повинна відповідати таким вимогам:

– висока продуктивність при обробці великої кількості предметів (понад 20 000 записів у каталозі);
– масштабованість архітектури для можливого розширення функціоналу;
– зручність користувацького інтерфейсу та інтуїтивна навігація;
– коректна обробка помилок та стабільність роботи API.

Реалізація зазначених задач дозволить створити повноцінну систему аналізу інвестиційних портфелів, яка забезпечує користувачам інструменти для оцінки ефективності вкладень у цифрові активи та прийняття обґрунтованих інвестиційних рішень.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Аналіз та вибір архітектури

Для реалізації програмної системи аналізу цифрових активів було обрано клієнт-серверну багаторівневу архітектуру, яка забезпечує розподіл відповідальності між користувацьким інтерфейсом, серверною логікою та базою даних [13].

Клієнтська частина системи реалізована з використанням фреймворку Angular та відповідає за відображення інтерфейсу користувача, обробку взаємодії з користувачем та надсилання HTTPS-запитів до серверної частини.

Серверна частина побудована на основі ASP.NET Core Web API та реалізує бізнес-логіку системи, обробку запитів, виконання розрахунків та взаємодію з базою даних [1].

Для збереження даних використовується система керування базами даних Microsoft SQL Server, яка забезпечує надійне зберігання інформації про користувачів, портфелі, цифрові предмети та історію їх вартості.

Взаємодія між компонентами здійснюється за допомогою REST API через HTTPS-запити, де клієнтська частина отримує та передає дані у форматі JSON [5]. Така архітектура забезпечує масштабованість, гнучкість та зручність подальшого розвитку системи.

2.2 Діаграми системи

2.2.1 Діаграма варіантів використання

Для формалізації функціональних можливостей системи побудовано діаграму варіантів використання, яка відображає взаємодію користувача та системи, а також фоновий процес оновлення ринкових даних.

На рисунку 2.1 представлено діаграму варіантів використання системи аналізу інвестиційного портфеля цифрових активів. Діаграма відображає основні функціональні можливості системи та взаємодію користувача із програмним забезпеченням.

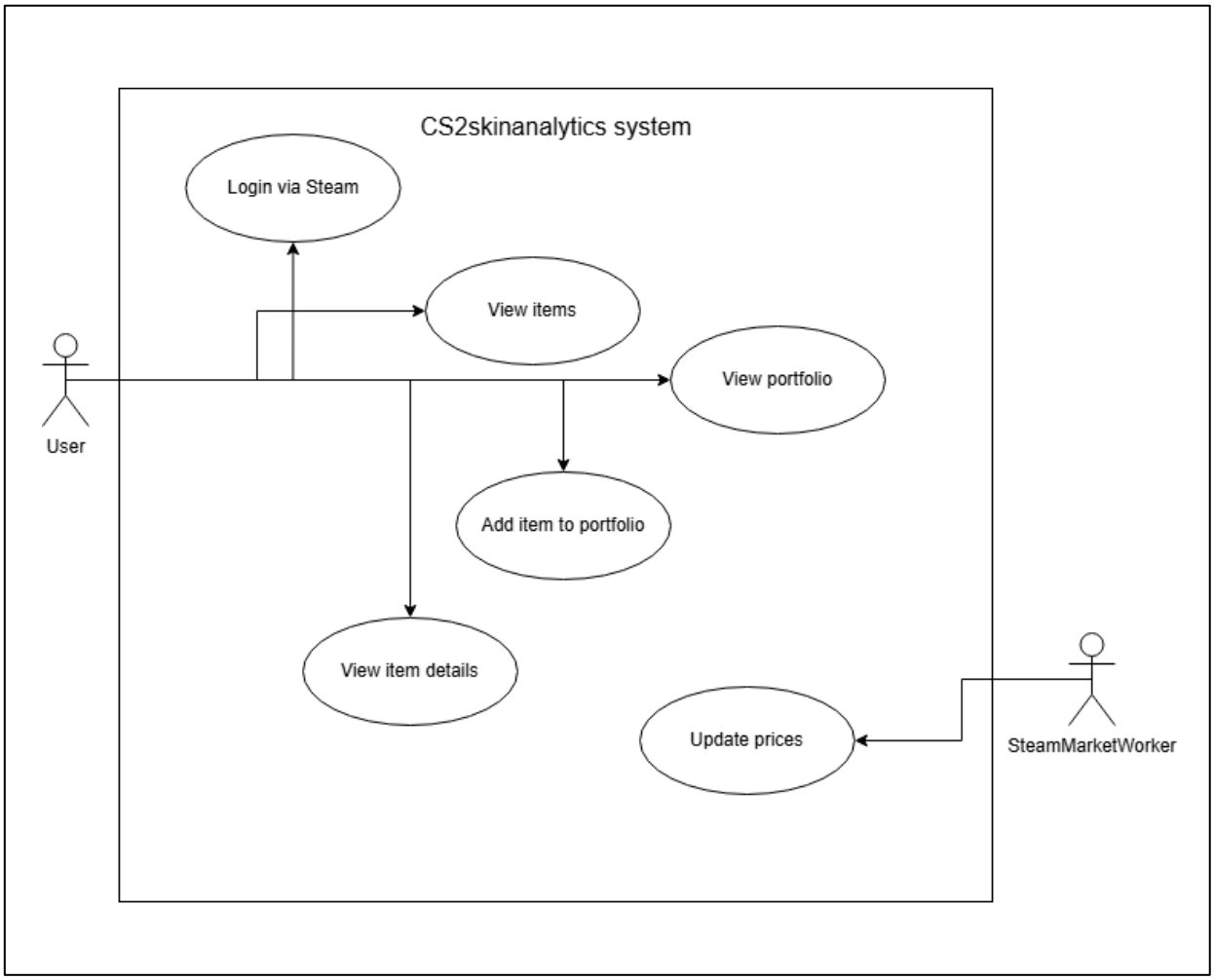


Рисунок 2.1 – Діаграма варіантів використання системи аналізу інвестиційного портфеля

Користувач може виконувати авторизацію через сервіс Steam, переглядати каталог предметів, отримувати детальну інформацію про окремі предмети, створювати та переглядати власні інвестиційні портфелі, а також додавати предмети до портфеля для подальшого аналізу.

Окремим актором представлено фоновий сервіс SteamMarketWorker, який автоматично виконує оновлення ринкових даних та забезпечує актуальність інформації про предмети в системі.

2.2.2 Діаграма класів

На рисунку 2.2 представлено діаграму класів програмної системи. Діаграма відображає основні класи предметної області, контролери, сервіси та взаємозв’язки між ними.

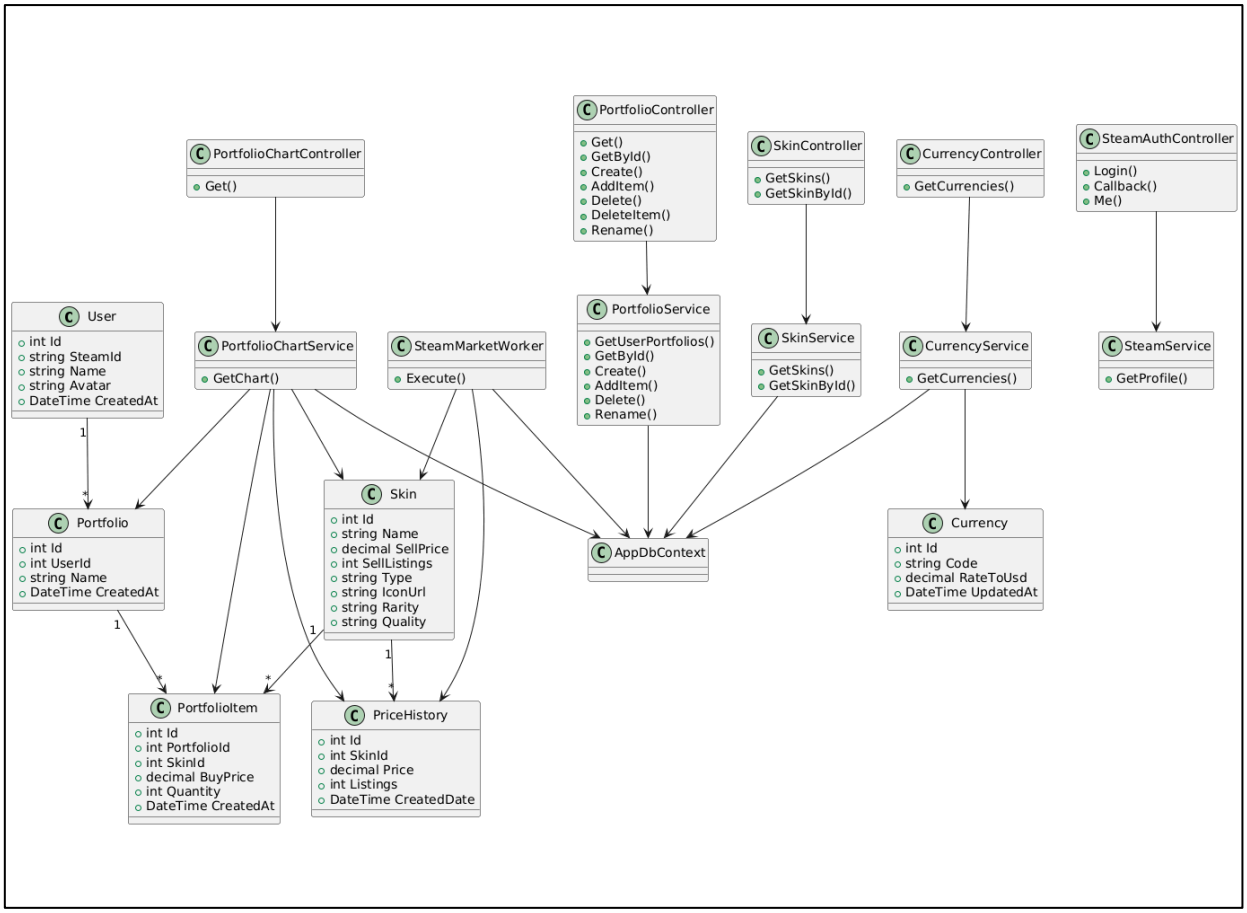


Рисунок 2.2 – Діаграма класів

До основних класів предметної області належать User, Portfolio, PortfolioItem, Skin, PriceHistory та Currency.

Для реалізації бізнес-логіки використовуються сервіси PortfolioService, PortfolioChartService, SkinService, CurrencyService та SteamService.

Обробка HTTP-запитів здійснюється відповідними контролерами, які взаємодіють із сервісами та контекстом бази даних AppDbContext [7].

Така структура забезпечує розділення відповідальності між компонентами системи та спрощує її подальшу підтримку й розвиток.

2.2.3 Діаграма компонентів

На рисунку 2.3 представлено компонентну діаграму системи, яка демонструє архітектуру взаємодії між основними програмними модулями.

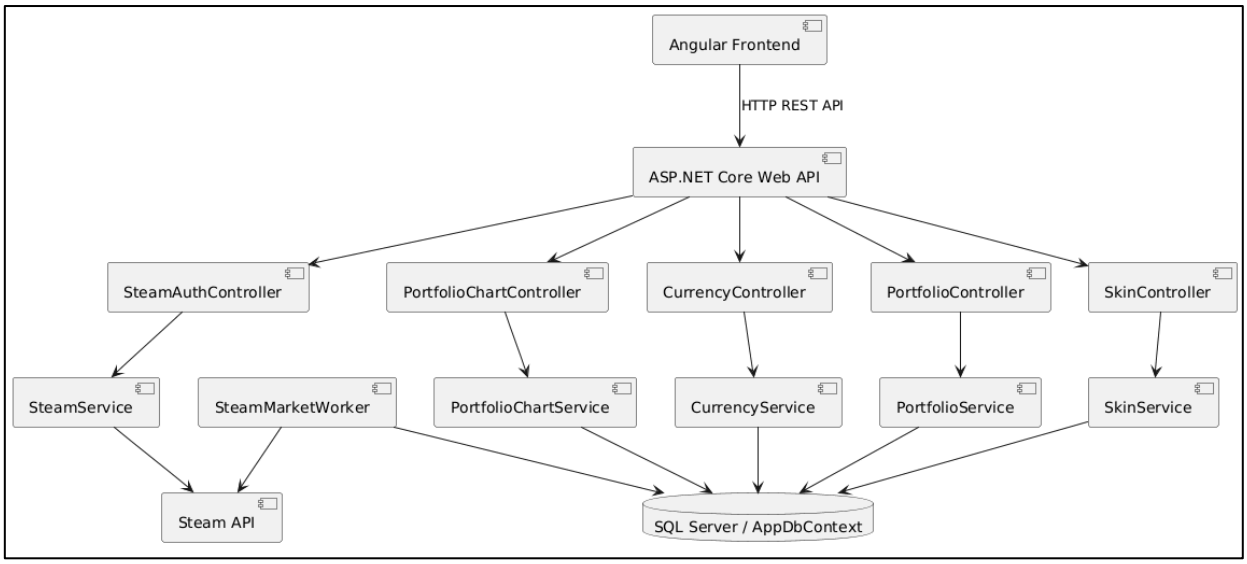


Рисунок 2.3 – Діаграма компонентів

Клієнтська частина реалізована на основі фреймворку Angular та взаємодіє із серверною частиною через REST API. Серверна частина побудована на платформі ASP.NET Core Web API та містить контролери, що забезпечують доступ до функціональних можливостей системи. Бізнес-логіка реалізована у відповідних сервісах, які взаємодіють із базою даних Microsoft SQL Server через контекст AppDbContext [3]. Для отримання інформації про користувачів та ринкові дані використовуються зовнішні сервіси Steam OpenID і Steam Market API. Запропонована архітектура забезпечує модульність, масштабованість та незалежність окремих компонентів системи.

2.2.4 Діаграми послідовностей

На рисунку 2.4 представлено діаграму послідовностей процесу авторизації користувача через сервіс Steam OpenID.

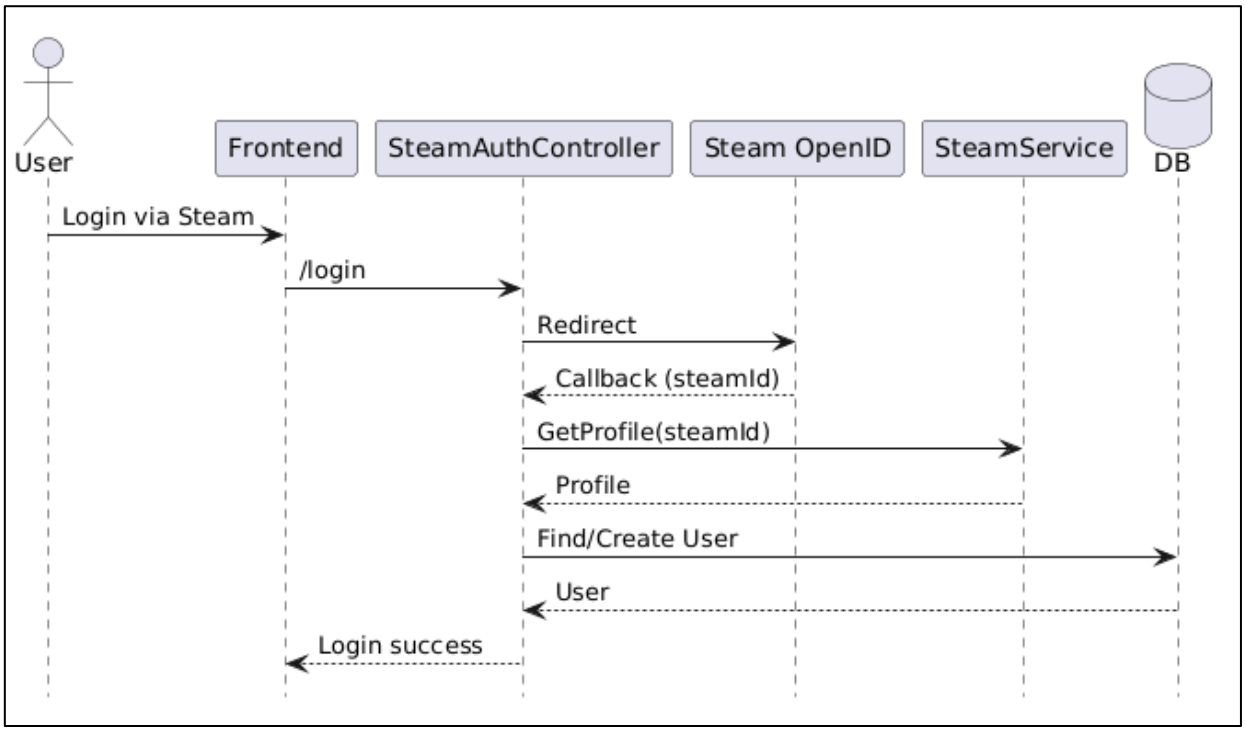


Рисунок 2.4 – Авторизація користувача через Steam

Процес розпочинається з ініціації входу користувачем через клієнтський інтерфейс. Після цього запит передається до контролера авторизації, який перенаправляє користувача на сторінку Steam OpenID. Після успішного підтвердження входу Steam повертає унікальний ідентифікатор користувача (SteamID). Далі система отримує інформацію про профіль користувача через Steam API, виконує пошук або створення запису в базі даних та формує авторизовану сесію. Після завершення процедури користувач отримує доступ до функціональних можливостей системи.

На рисунку 2.5 представлено діаграму послідовностей процесу додавання предмета до інвестиційного портфеля.

Користувач обирає предмет у клієнтському інтерфейсі та надсилає запит на його додавання до портфеля. Запит передається до PortfolioController, який викликає відповідний метод PortfolioService. Сервіс виконує перевірку коректності отриманих даних, створює новий запис у таблиці PortfolioItems та зберігає його у базі даних через AppDbContext. Після успішного виконання операції система повертає відповідь клієнту, а інформація про склад портфеля оновлюється в інтерфейсі користувача.

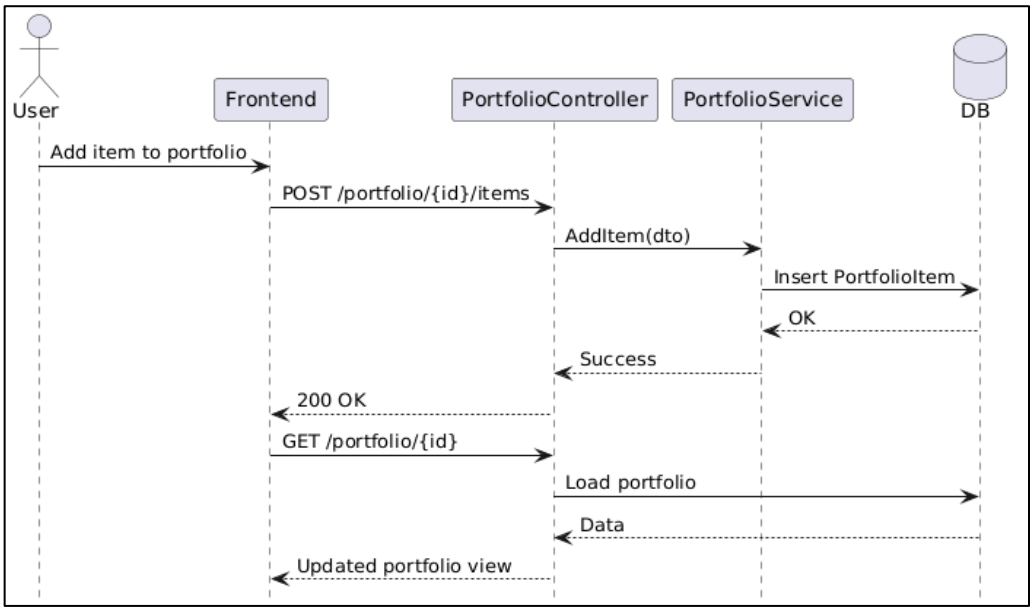


Рисунок 2.5 – Додавання предмета до портфеля

2.3 Проєктування бази даних

Для збереження та обробки даних у системі аналізу цифрових активів було спроєктовано реляційну базу даних, яка забезпечує ефективне зберігання інформації про користувачів, інвестиційні портфелі, цифрові предмети, валютні курси та історію змін вартості предметів.

Основними сутностями системи є Users, Portfolios, PortfolioItems, Skins, PriceHistory та Currencies.

Таблиця users містить дані про користувачів системи, включаючи їх Steam-ідентифікатор, ім’я та аватар. Кожен користувач може мати декілька інвестиційних портфелів, що реалізує зв’язок типу «один до багатьох» між Users та Portfolios.

Таблиця portfolios використовується для зберігання інвестиційних портфелів користувачів та містить загальну інформацію про портфель. Кожен портфель складається з набору позицій, що зберігаються у таблиці PortfolioItems.

Таблиця portfolioItems є ключовою сутністю системи та містить інформацію про придбані користувачем цифрові активи, включаючи кількість

предметів, ціну покупки та дату створення запису. Вона пов’язує портфелі з конкретними ігровими предметами, що дозволяє формувати структуру інвестицій користувача.

Таблиця skins використовується як довідкова таблиця та містить інформацію про цифрові предмети, включаючи назву, тип, рідкість, якість, зображення та поточну ринкову вартість. Один і той самий предмет може використовуватися в різних портфелях через зв’язок із PortfolioItems.

Таблиця pricehistory призначена для збереження історії зміни вартості цифрових предметів у часі, що дозволяє будувати графіки динаміки цін та виконувати аналітичні розрахунки.

Таблиця currencies використовується для збереження інформації про валютні курси та забезпечує можливість конвертації вартості активів у еквівалент долара США.

На рисунку 2.6 представлено ER-діаграму бази даних системи аналізу цифрових активів.

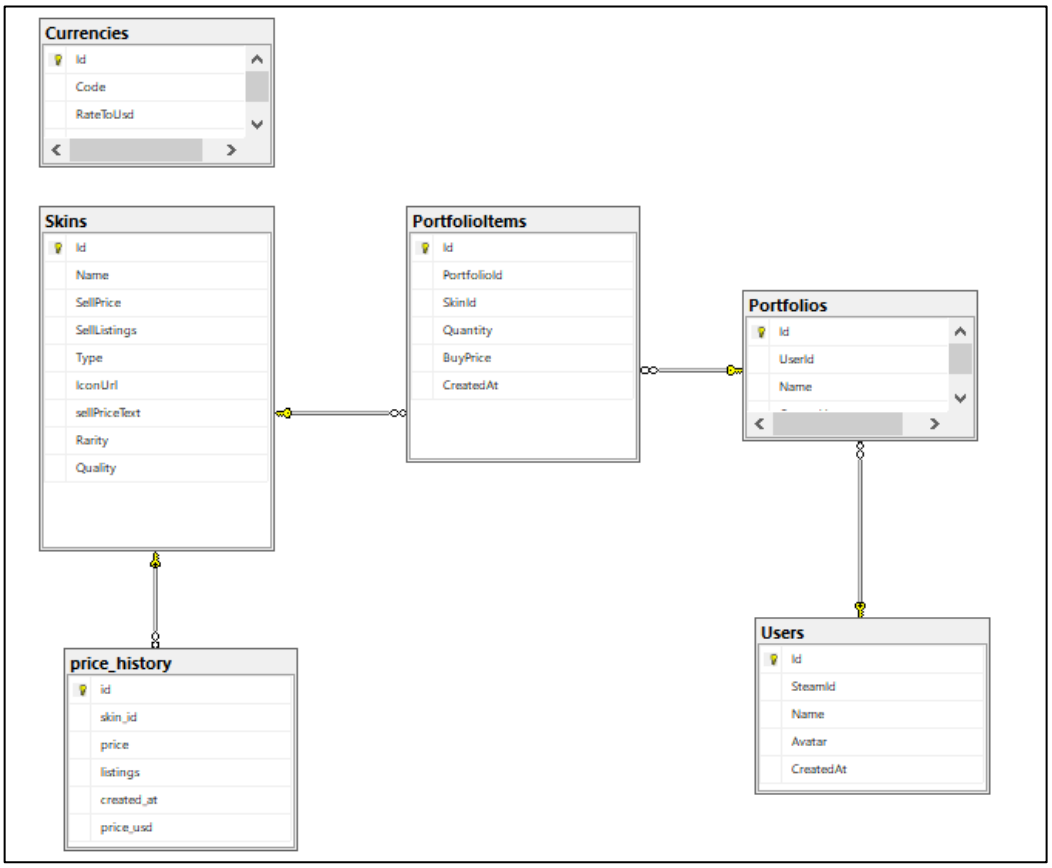


Рисунок 2.6 – ER-діаграма бази даних системи

Спроєктована структура бази даних забезпечує нормалізацію даних, мінімізацію дублювання інформації та ефективну обробку великих обсягів даних, що є критично важливим для аналітичної системи цифрових активів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Реалізація авторизації та збору даних

3.1.1 Авторизація через Steam OpenID

Для авторизації користувачів у системі використовується технологія Steam OpenID [9]. Даний механізм дозволяє виконувати автентифікацію без необхідності створення окремих облікових записів у системі.

Після успішного входу користувач отримує унікальний ідентифікатор SteamId, який використовується для подальшої взаємодії із системою.

Для отримання додаткової інформації про користувача використовується Steam Web API [8]. Система отримує ім’я користувача та посилання на аватар профілю, після чого ці дані зберігаються у таблиці Users.

На рисунках 3.1 та 3.2 послідовно представлено фрагмент серверної логіки отримання інформації про користувача через Steam API.

```
[HttpGet("login")]
public IActionResult Login()
{
    return Challenge(new AuthenticationProperties
    {
        RedirectUri = "api/auth/steam/callback"
    }, "Steam");
}

[HttpGet("callback")]
public async Task<IActionResult> Callback()
{

    var result = await HttpContext.AuthenticateAsync("Steam");

    if (!result.Succeeded || result.Principal == null)
        return Unauthorized("Steam auth failed");

    var claimedId = result.Principal.FindFirst(ClaimTypes.NameIdentifier)?.Value;

    if (claimedId == null)
        return Unauthorized("No steam id");

    var steamId = claimedId.Split('/').Last();

    var user = _db.Users.FirstOrDefault(x => x.SteamId == steamId);

    if (user == null)
    {
        var apiKey = _config["Steam:ApiKey"];
        var profile = await _steamService.GetProfileAsync(steamId, apiKey);

        user = new User
        {
```

Рисунок 3.1 – Метод авторизації через Steam OpenID (1 частина)



```
            user = new User
            {
                SteamId = steamId,
                Name = profile.name,
                Avatar = profile.avatar
            };

            _db.Users.Add(user);
            await _db.SaveChangesAsync();
        }

        var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, steamId)
    };

        var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
        var principal = new ClaimsPrincipal(identity);

        await HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme,
            principal,
            new AuthenticationProperties
            {
                IsPersistent = true,
                ExpiresUtc = DateTimeOffset.UtcNow.AddDays(7)
            }
        );

        return Redirect("                    :4200");
```

Рисунок 3.2 – Метод авторизації через Steam OpenID (2 частина)

Отримані дані використовуються для створення нового запису користувача у базі даних.

3.1.2 Реалізація системи збору даних про предмети

Для реалізації автоматичного збору інформації використовується фоновий сервіс SteamMarketWorker, створений на базі BackgroundService платформи ASP.NET Core [6, 15]. Даний сервіс працює незалежно від основного потоку вебзастосунку та виконує періодичні HTTP-запити до Steam Market API [10]. На рисунках 3.3 та 3.4 послідовно представлено реалізацію фонового сервісу збору даних.



```
public class SteamMarketWorker : BackgroundService
{
    private readonly IServiceScopeFactory _scopeFactory;
    private readonly HttpClient _http;

    private const string BaseUrl =
        "https://steamcommunity.com/market/search/render/?query=&appid=730&norender=1&currency=0&count=10&start=";

    Ссылок: 0
    public SteamMarketWorker(
        IServiceScopeFactory scopeFactory,
        IHttpClientFactory httpFactory)
```

Рисунок 3.3 – Фоновий сервіс збору даних (1 частина)

```
        IHttpClientFactory httpFactory)
    {

        _scopeFactory = scopeFactory;
        _http = httpFactory.CreateClient("Steam");

        _http.DefaultRequestHeaders.Add(
            "User-Agent",
            "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 Chrome/139.0.0.0 Safari/537.36");

        _http.DefaultRequestHeaders.Add("Accept", "*/*");
        _http.DefaultRequestHeaders.Add("Referer", "https://steamcommunity.com/market/");
        _http.DefaultRequestHeaders.Add("Accept-Language", "en-US,en;q=0.9");
    }

    Ссылок: 0
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        int start = 0;
        int totalCount = int.MaxValue;

        var rnd = new Random();

        while (!stoppingToken.IsCancellationRequested)
        {
            using var scope = _scopeFactory.CreateScope();

            var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();

            var url = BaseUrl + start;

            try
            {
                Console.WriteLine($"REQUEST: {url}");

                var response = await _http.GetStringAsync(url, stoppingToken);

                using var json = JsonDocument.Parse(response);

                var root = json.RootElement;

                if (root.TryGetProperty("total_count", out var total))
```

Рисунок 3.4 – Фоновий сервіс збору даних (2 частина)

Після отримання відповіді виконується її обробка за допомогою класу JsonDocument. Із JSON-структури система отримує такі параметри:

– назву предмета (hash_name);
– ринкову ціну (sell_price);
– кількість активних лотів (sell_listings);
– URL зображення предмета (icon_url).

На рисунку 3.5 представлено процес обробки JSON-відповіді Steam Market API.

Після отримання даних виконується перевірка наявності предмета у базі даних.

Якщо предмет відсутній, створюється новий запис у таблиці Skins. Якщо запис вже існує, система оновлює його поточні параметри, зокрема ринкову ціну, кількість лотів та URL зображення.

```
var root = json.RootElement;

if (root.TryGetProperty("total_count", out var total))
{
    totalCount = total.GetInt32();
}

var uahCurrency = await db.Currencies
    .AsNoTracking()
    .FirstOrDefaultAsync(x => x.Code == "UAH", stoppingToken);

if (root.TryGetProperty("results", out var results))
{
    foreach (var item in results.EnumerateArray())
    {
        var hashName = item.GetProperty("hash_name").GetString();

        if (string.IsNullOrWhiteSpace(hashName))
            continue;

        string skinName = hashName;
        string? quality = null;
        string? rarity = null;
        string? type = null;

        var qualities = new[]
        {
            "Factory New",
            "Minimal Wear",
            "Field-Tested",
            "Well-Worn",
            "Battle-Scarred"
        };

        foreach (var q in qualities)
        {
            var suffix = $"({q})";

            if (skinName.EndsWith(suffix))
            {
                quality = q;
                skinName = skinName.Replace($" {suffix}", "");
                break;
            }
        }

        int sellListings = 0;

        if (item.TryGetProperty("sell_listings", out var listingsElement))
        {
            sellListings = listingsElement.GetInt32();
        }

        decimal sellPrice = 0;
        string? sellPriceText = null;
        string? iconUrl = null;

        if (item.TryGetProperty("sell_price_text", out var priceTextElement))
        {
            sellPriceText = priceTextElement.GetString();
        }

        if (item.TryGetProperty("asset_description", out var assetDescription))
        {
            if (assetDescription.TryGetProperty("icon_url", out var iconElement))
            {
                iconUrl = iconElement.GetString();
            }
```

Рисунок 3.5 – Процес обробки JSON-відповіді Steam Market API

Для роботи з базою даних використовується технологія Entity Framework Core та контекст AppDbContext [2].

На рисунку 3.6 представлено процес оновлення інформації про предмети у базі даних.

Реалізований механізм автоматичного збору даних дозволяє підтримувати актуальність інформації про предмети CS2 та забезпечує автоматичне оновлення ринкових даних у системі аналітики інвестиційного портфеля.

```
    Console.WriteLine(
        $"{skinName} | {quality} | {rarity} | {type} | {sellPrice}$");

    var existingSkin = await db.Skins
        .FirstOrDefaultAsync(
            x => x.Name == skinName &&
                 x.Quality == quality,
            stoppingToken);

    if (existingSkin == null)
    {
        db.Skins.Add(new Skin
        {
            Name = skinName,
            Quality = quality,
            Type = type,
            Rarity = rarity,
            SellPrice = sellPrice,
            SellPriceText = sellPriceText,
            SellListings = sellListings,
            IconUrl = iconUrl
        });

        Console.WriteLine(
            $"ADDED: {skinName} [{quality}]");
    }
    else
    {
        existingSkin.SellPrice = sellPrice;
        existingSkin.SellPriceText = sellPriceText;
        existingSkin.SellListings = sellListings;
        existingSkin.Type = type;
        existingSkin.Rarity = rarity;

        if (!string.IsNullOrWhiteSpace(iconUrl))
        {
            existingSkin.IconUrl = iconUrl;
        }

        Console.WriteLine(
            $"UPDATED: {skinName} [{quality}]");
    }
}

var changes = await db.SaveChangesAsync(stoppingToken);

Console.WriteLine($"DB CHANGES: {changes}");
```

Рисунок 3.6 – Процес оновлення інформації про предмети у базі даних

3.2 Реалізація аналітики інвестиційного портфеля

3.2.1 Розрахунок PnL (Profit and Loss)

PnL (Profit and Loss) є одним із ключових показників інвестиційної аналітики, який відображає фінансовий результат користувача від володіння предметами CS2. Даний показник визначає різницю між поточною ринковою вартістю предмета та його ціною придбання.

У розробленій системі PnL розраховується для кожного елемента портфеля окремо, після чого може агрегуватися для отримання загального результату.

Базова логіка розрахунку має вигляд:

– якщо поточна ціна більша за ціну покупки – користувач отримує прибуток;
– якщо менша – фіксується збиток.

Реалізація розрахунку виконується на стороні серверної логіки або клієнтської частини шляхом обробки даних портфеля.

На рисунку 3.7 представлено реалізацію алгоритму розрахунку PnL.

```
var totalInvested = portfolio.Items.Sum(i => i.BuyPrice * i.Quantity);
var totalValue = portfolio.Items.Sum(i => i.Skin.SellPrice * i.Quantity);
var totalProfit = totalValue - totalInvested;
```

Рисунок 3.7 – Алгоритм розрахунку PnL

3.2.2 Розрахунок ROI (Return on Investment)

ROI (Return on Investment) використовується для оцінки ефективності інвестицій та показує відносну прибутковість вкладень у відсотковому вираженні.

У системі, ROI обчислюється на основі співвідношення прибутку до суми інвестицій за формулою.

ROI демонструє наскільки ефективно було вкладено кошти у конкретний предмет або весь портфель.

Даний показник дозволяє користувачу швидко оцінити доцільність інвестицій у певні скіни та порівнювати різні активи між собою.

Розрахунок виконується динамічно на основі актуальних даних про ринкову вартість предметів.

На рисунку 3.8 представлено реалізацію алгоритму розрахунку ROI.

```
var roi = invested == 0
    ? 0
    : (profit / invested) * 100;
```

Рисунок 3.8 – Алгоритм розрахунку ROI

3.2.3 Візуалізація аналітичних даних (графіки портфеля)

В рамках інформаційної системи реалізовано візуалізацію аналітичних показників портфеля користувача у вигляді графіків. Це дозволяє відстежувати динаміку інвестицій у часі та швидко оцінювати ефективність портфеля.

На рисунку 3.9 алгоритм побудови часових даних для графіку портфеля.

```
for (var date = fromDate; date <= toDate; date = date.AddDays(1))
{
    decimal totalValue = 0;
    decimal invested = 0;

    foreach (var item in portfolio)
    {

        if (item.CreatedAt.Date > date)
            continue;

        invested += item.BuyPrice * item.Quantity;

        if (!priceLookup.TryGetValue(item.SkinId, out var history))
            continue;

        var price = history
            .LastOrDefault(x => x.CreatedDate.Date <= date);

        if (price == null)
            continue;

        totalValue += price.Price * item.Quantity;
    }

    var pnl = totalValue - invested;

    result.Add(new PortfolioChartPointDto
    {
        Date = date,
        Value = totalValue,
        PnL = pnl
    });
}

return new PortfolioChartResponseDto
{
    Data = result,
    ChangePercent7d = Calc(result, 7),
    ChangePercent30d = Calc(result, 30),
    ChangePercent1y = Calc(result, 365),
    ChangePercentAll = SafePercent(result)
};
```

Рисунок 3.9 – Алгоритм побудови часових даних для графіку портфеля

Алгоритм виконує покрокову симуляцію зміни вартості портфеля в часовому розрізі. для кожного дня обчислюються загальна ринкова вартість активів та вкладені інвестиції на основі історичних цін, після чого формується показник прибутку (pnl). отримані дані зберігаються у вигляді точок графіку для подальшої візуалізації.

3.3 Реалізація користувацького інтерфейсу

Користувацький інтерфейс вебзастосунку реалізовано з використанням фреймворку Angular, що забезпечує побудову динамічного, адаптивного та компонентного інтерфейсу [4]. Основною метою UI є забезпечення зручної взаємодії користувача з інвестиційною аналітичною системою та надання доступу до ключових функцій у зрозумілому вигляді.

Архітектура клієнтської частини побудована на основі компонентного підходу, де кожна сторінка є окремим незалежним компонентом. Такий підхід дозволяє легко масштабувати систему та повторно використовувати логіку між різними частинами застосунку.

3.3.1 Головна сторінка

Головна сторінка вебзастосунку призначена для забезпечення користувача доступом до основних функціональних можливостей системи. Саме з даної сторінки починається взаємодія користувача із програмним продуктом.

У якості головної сторінки реалізовано каталог предметів платформи Steam для гри Counter-Strike 2. Каталог містить інформацію про понад 25 тисяч предметів, що доступні для перегляду та аналізу. Дані про предмети автоматично завантажуються із зовнішнього джерела та зберігаються у базі даних системи.

Для спрощення пошуку необхідного предмета на сторінці реалізовано пошукове поле, яке дозволяє здійснювати пошук за назвою предмета. Також передбачено систему фільтрації, що дає можливість відображати предмети відповідно до обраних категорій, типів зброї або інших характеристик.

Кожен елемент списку містить основну інформацію про предмет, зокрема його назву, поточну ринкову вартість, кількість активних лотів на

торговому майданчику та графічне зображення. При натисканні на картку предмета користувач переходить до сторінки детального перегляду.

На рисунку 3.10 наведено зовнішній вигляд головної сторінки вебзастосунку.

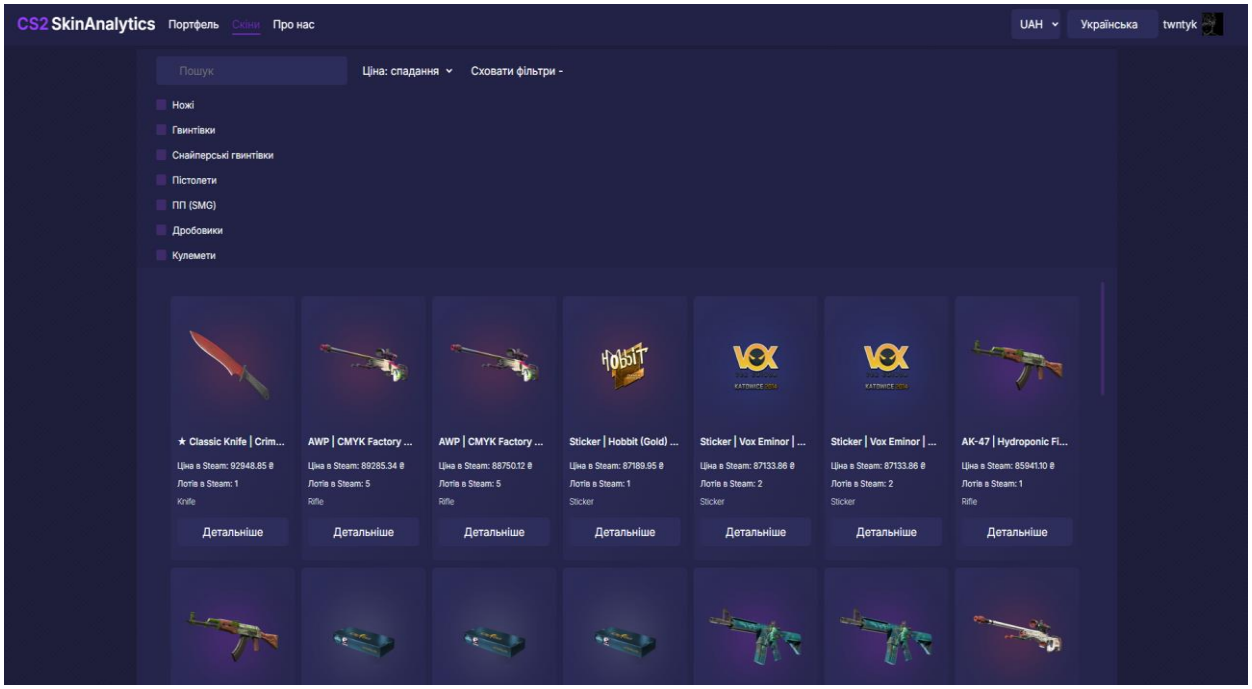


Рисунок 3.10 – Головна сторінка вебзастосунку з каталогом предметів та системою пошуку і фільтрації

3.3.2 Сторінка перегляду предмета

На цій сторінці користувач може отримати детальну інформацію про обраний предмет. Сторінка формується на основі даних, отриманих із бази даних, та містить основні характеристики предмета, поточну ринкову вартість, кількість активних лотів на торговельному майданчику Steam, а також зображення предмета.

Одним із ключових елементів сторінки є графік зміни вартості предмета, побудований на основі історичних даних про ціни. Графік дозволяє користувачу аналізувати динаміку зміни вартості предмета за різні часові проміжки та оцінювати доцільність інвестування.

Також на сторінці реалізовано можливість переходу до офіційної сторінки предмета на торговельному майданчику Steam для отримання додаткової інформації та виконання операцій купівлі або продажу.

На рисунку 3.11 наведено сторінку перегляду предмета.



Рисунок 3.11 – Сторінка перегляду предмета

3.3.3 Авторизація користувача через Steam

Для забезпечення безпечної ідентифікації користувачів у вебзастосунку реалізовано механізм авторизації через сервіс Steam OpenID. Використання даного підходу дозволяє уникнути необхідності створення окремої системи реєстрації та зберігання паролів, оскільки автентифікація виконується засобами платформи Steam.

Процес авторизації розпочинається після натискання користувачем кнопки входу через Steam. Після цього система перенаправляє користувача на сторінку авторизації Steam, де він підтверджує вхід до свого облікового запису. Після успішної перевірки Steam повертає унікальний ідентифікатор користувача (SteamID), який використовується для подальшої роботи із системою.

Під час першої авторизації застосунок додатково отримує через Steam Web API інформацію про профіль користувача, зокрема ім’я та аватар. Отримані дані зберігаються у базі даних для подальшого використання. При наступних входах система виконує пошук користувача за SteamID та створює авторизовану сесію без повторного збереження даних.

Завдяки використанню Steam OpenID користувач отримує швидкий доступ до функціоналу вебзастосунку, а система забезпечує надійну ідентифікацію без необхідності зберігати конфіденційні облікові дані.

На рисунку 3.12 наведено сторінку авторизації користувача через Steam.

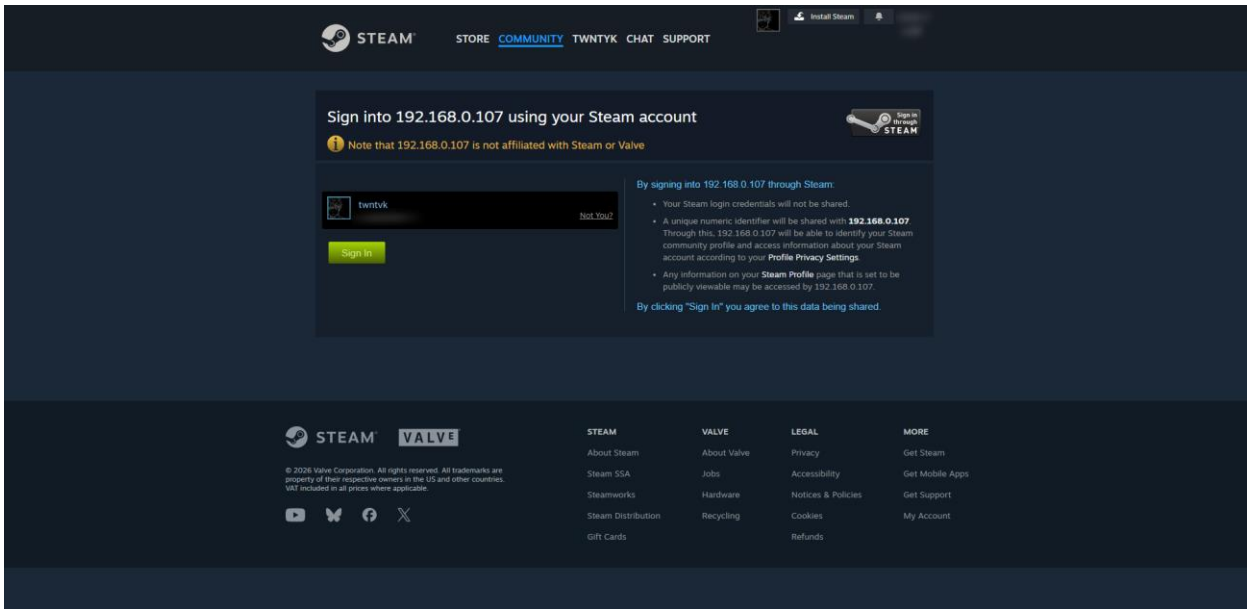


Рисунок 3.12 – Сторінка авторизації користувача через Steam

3.3.4 Сторінка портфелів користувача

Після успішної авторизації користувач отримує доступ до сторінки портфелів, яка призначена для управління інвестиціями у цифрові активи. На даній сторінці відображається список усіх створених портфелів користувача із основною інформацією щодо їх поточного стану.

Для кожного портфеля відображаються його назва, дата створення, поточна ринкова вартість активів (Portfolio Value), загальний прибуток або

збиток (PnL), а також показник рентабельності інвестицій (ROI). Це дозволяє користувачу швидко оцінити ефективність власних інвестицій без необхідності переходу до детального перегляду портфеля.

Також на сторінці реалізовано функції створення нового портфеля, перейменування існуючих портфелів та їх видалення. Для переходу до детальної інформації про склад портфеля користувач може скористатися кнопкою відкриття відповідного портфеля.

Завдяки компактному представленню основних показників сторінка забезпечує зручний моніторинг стану всіх інвестиційних портфелів користувача в одному місці.

На рисунку 3.13 наведено сторінку інвестиційних портфелів користувача.

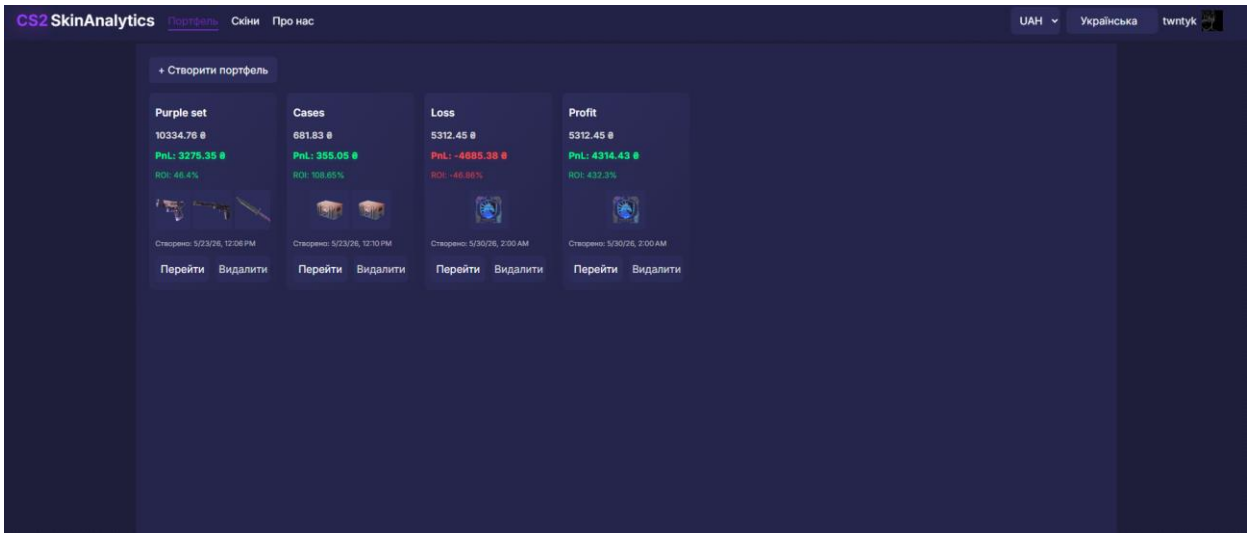


Рисунок 3.13 – Сторінка інвестиційних портфелів користувача

3.3.5 Сторінка детального перегляду портфеля

Сторінка детального перегляду портфеля призначена для аналізу складу інвестиційного портфеля та оцінки ефективності вкладень користувача. На сторінці відображається перелік усіх предметів, що входять до складу обраного портфеля, а також фінансові показники для кожної позиції.

Для кожного предмета відображаються його назва, кількість, ціна придбання, поточна ринкова вартість, прибуток або збиток (PnL) та показник рентабельності інвестицій (ROI).

Предмети можуть бути згруповані за назвою та якістю, що дозволяє зручно працювати з великими обсягами даних.

У верхній частині сторінки розташовано узагальнену інформацію про портфель, зокрема загальну ринкову вартість активів, сумарний прибуток або збиток та показник ROI для всього портфеля. Це дає можливість швидко оцінити поточний стан інвестицій.

Крім табличного представлення даних, сторінка містить графік зміни вартості портфеля у часі [14]. Графік будується на основі історичних даних про ціни предметів та дозволяє відстежувати динаміку зміни вартості інвестицій за різні періоди часу.

Також реалізовано функціонал додавання нових предметів до портфеля та видалення існуючих позицій, що забезпечує можливість актуалізації складу інвестицій відповідно до змін на ринку.

На рисунку 3.14 наведено сторінку детального перегляду інвестиційного портфеля користувача.

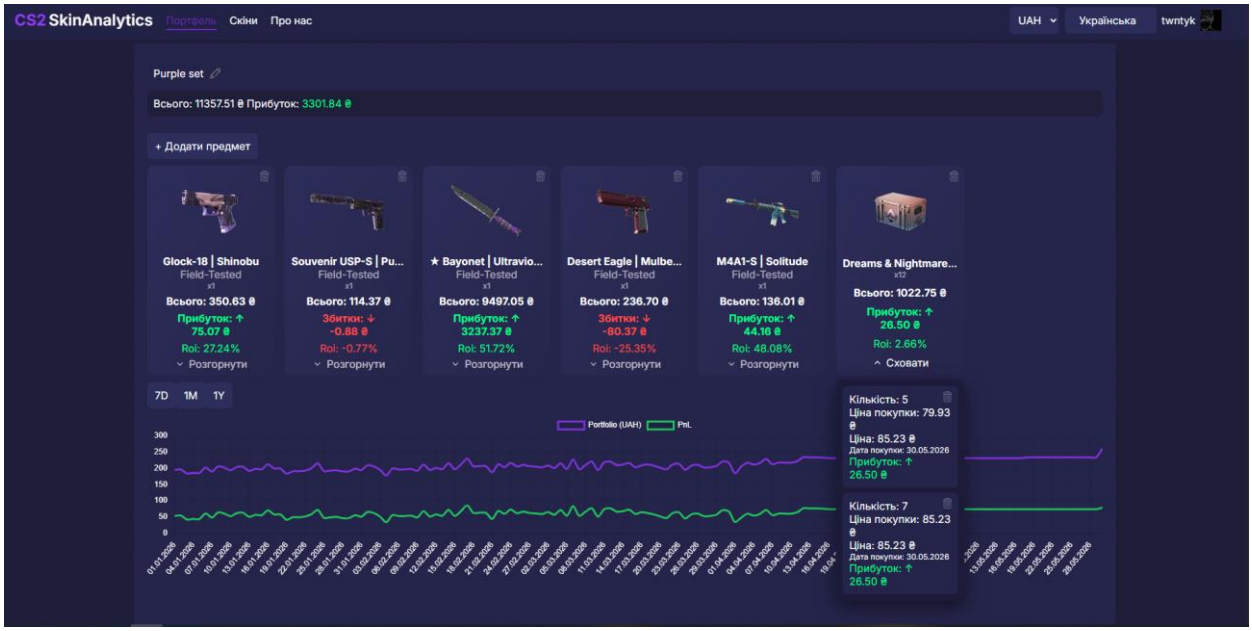


Рисунок 3.14 – Сторінка детального перегляду портфеля

3.4 Тестування серверної частини та опис API

Під час розробки вебзастосунку було виконано тестування серверної частини системи, що включає REST API, бізнес-логіку, взаємодію з базою даних та механізми авторизації. Основною метою тестування було перевірити коректність роботи всіх функціональних модулів та стабільність обробки запитів.

Серверна частина системи реалізована на основі ASP.NET Core Web API, що забезпечує обробку HTTPS-запитів та повернення даних у форматі JSON. Взаємодія з клієнтською частиною здійснюється через HTTPS-протокол, що гарантує безпечну передачу даних.

У межах системи реалізовано групи API-ендпоінтів, які розглянемо далі.

3.4.1 Авторизація користувача

Для автентифікації користувачів використовується Steam OpenID, що дозволяє виконувати вхід без створення окремих облікових записів у системі.

Основні методи:

– GET /api/auth/steam/login – ініціація авторизації через Steam;
– GET /api/auth/steam/callback – обробка відповіді Steam та створення користувача;
– GET /api/auth/steam/me – отримання даних поточного користувача.

Після успішної авторизації користувач отримує cookie-сесію, яка використовується для доступу до захищених ресурсів системи.

3.4.2 Управління портфелями

Даний модуль відповідає за створення та управління інвестиційними портфелями користувачів.

Основні методи:

– GET /api/portfolio – отримання всіх портфелів користувача;
– POST /api/portfolio – створення нового портфеля;
– GET /api/portfolio/{id} – детальна інформація про портфель;
– DELETE /api/portfolio/{id} – видалення портфеля;
– PUT /api/portfolio/{id}/rename – перейменування портфеля;
– POST /api/portfolio/{id}/items – додавання предмета до портфеля;
– DELETE /api/portfolio/items/{itemId} – видалення предмета з портфеля.

Усі операції виконуються з перевіркою належності портфеля авторизованому користувачу.

3.4.3 Аналітика портфеля

Для побудови графіків зміни вартості портфеля використовується спеціальний аналітичний API.

GET /api/portfolio-chart – отримання часових даних для побудови графіка. Результатом є набір точок, що містять:

– дату;
– загальну вартість портфеля;
– значення прибутку або збитку (PnL).

Ці дані використовуються для побудови графіків динаміки інвестицій. Операція виконується з перевіркою належності портфеля авторизованому користувачу.

3.4.4 Робота з предметами

Дані методи які використовуються для відображення каталогу предметів та сторінки перегляду конкретного активу.

Основні методи:

– GET /api/Skin – отримання списку всіх предметів;
– GET /api/Skin/{id} – детальна інформація про предмет.

3.4.5 Додаткові сервіси системи

Додаткові методи які використовуються для отримання допоміжних даних:

– GET /api/Currency – отримання актуальних курсів валют;
– GET /api/stats – отримання загальної статистики системи;
– GET /api/chart – допоміжні аналітичні дані.

3.4.6 Безпека API та захист від типових вебвразливостей

Під час розробки серверної частини системи було враховано основні принципи захисту вебзастосунків та виконано аналіз на наявність типових вразливостей, таких як SQL-ін’єкції, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), а також несанкціонований доступ до API.

Завдяки використанню ORM Entity Framework Core всі запити до бази даних формуються параметризовано, що виключає можливість SQL-ін’єкцій.

Для взаємодії між клієнтською та серверною частинами використовується формат JSON, що мінімізує ризики XSS-атак на серверному рівні. Додатково Angular-клієнт автоматично екранує динамічний контент при відображенні в інтерфейсі.

Механізм авторизації побудований на основі cookie-сесій та Steam OpenID, що забезпечує захист від несанкціонованого доступу. Усі запити до захищених API перевіряють наявність дійсної сесії користувача та відповідність UserId, що виключає доступ до чужих даних.

Для зниження ризику CSRF-атак використовується модель, заснована на cookie-аутентифікації в межах одного домену, а також обмежений доступ до критичних операцій лише для авторизованих користувачів.

Додатково реалізовано контроль доступу на рівні бізнес-логіки, що запобігає виконанню операцій над ресурсами, які не належать поточному користувачу.

Таким чином, у межах реалізації системи було забезпечено базовий рівень захисту від основних класів вебвразливостей, характерних для сучасних клієнт-серверних застосунків.

3.4.7 Результати тестування

Під час тестування серверної частини вебзастосунку було перевірено коректність роботи всіх реалізованих API-ендпоінтів, бізнес-логіки та взаємодії з базою даних.

Основна увага приділялась перевірці CRUD-операцій для інвестиційних портфелів, роботі механізму авторизації через Steam OpenID, а також коректності розрахунку аналітичних показників (PnL, ROI та загальної вартості портфеля).

Тестування виконувалось із використанням інструментів Swagger UI та HTTPS-клієнтів, що дозволило імітувати запити від клієнтської частини та перевірити формування відповідей у форматі JSON.

У ході перевірки було встановлено, що:

– усі захищені ендпоінти коректно обмежують доступ неавторизованим користувачам;
– операції створення, оновлення та видалення портфелів виконуються без помилок;
– розрахунки фінансових показників виконуються коректно на основі актуальних даних;

- система стабільно обробляє запити до зовнішніх сервісів (Steam Market API);
- механізм авторизації забезпечує коректне створення та збереження користувацьких сесій.

Отримані результати тестування підтверджують стабільну роботу серверної частини системи та відповідність реалізованої функціональності вимогам технічного завдання.

ВИСНОВКИ

У результаті виконання дипломної роботи мною було розроблено веборієнтовану систему для аналізу інвестиційних портфелів цифрових активів у грі Counter-Strike 2. Розроблене програмне забезпечення виконує автоматичний збір даних предметів, зберігання інформації про предмети та портфелі користувачів, також було зроблено систему розрахунку аналітичних показників і візуалізацію результатів у зручному для користувача вигляді.

Розроблена система побудована із використанням Angular, ASP.NET Core та Microsoft SQL Server, це забезпечує модульність, та масштабованість також можливість подальшого розвитку. Архітектурні рішення дозволяють розширювати функціональність у найбільш ефективному форматі.

Використання Steam OpenID для автентифікації користувачів та контроль доступу забезпечує вискоий рівень безпеки даних. Застосування Entity Framework Core та параметрів у запитах дозволяє обійти ризики виникнення типових вразливостей, які притаманні роботи із базою даних.

Алгоритми розрахунку PnL, ROI та побудови графіків на основі історичних даних дозволяють користувачам отримувати актуальну інформацію про стан інвестицій та приймати більш обґрунтовані рішення щодо управління цифровими активами.

Проведене тестування підтвердило працездатність, стабільність роботи розробленої системи. Отримані результати кажуть про досягнення поставленої мети роботи та можливість практичного використання створеного мною програмного забезпечення для моніторингу й аналізу цифрових активів користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. ASP.NET Core Documentation. *Microsoft*. URL: https://learn.microsoft.com/aspnet/core (дата звернення: 13.03.2026).

2. Entity Framework Core Documentation. *Microsoft*. URL: https://learn.microsoft.com/ef/core (дата звернення: 15.03.2026).

3. SQL Server Documentation. *Microsoft*. URL: https://learn.microsoft.com/sql/sql-server (дата звернення: 15.03.2026).

4. Angular Documentation. *Official Angular Guide*. URL: https://angular.dev/overview (дата звернення: 17.03.2026).

5. REST API Tutorial. URL: https://restfulapi.net (дата звернення: 18.03.2026).

6. Dependency Injection in .NET. *Microsoft*. URL: https://learn.microsoft.com/dotnet/core/extensions/dependency-injection (дата звернення: 21.03.2026).

7. HTTP Overview. *Mozilla Developer Network*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (дата звернення: 21.03.2026).

8. Steam Web API Documentation. *Steam*. URL: https://developer.valvesoftware.com/wiki/Steam_Web_API (дата звернення: 30.04.2026).

9. OpenID Authentication. *Steam*. URL: https://partner.steamgames.com/doc/features/auth?l=english (дата звернення: 28.04.2026).

10. Valve Corporation. *Steam Community Market API*. URL: https://steamcommunity.com/market?l=english (дата звернення: 29.04.2026).

11. Profit and Loss (PnL) Definition. *Investopedia*. URL: https://www.investopedia.com/terms/p/plstatement.asp (дата звернення: 07.05.2026).

12. Return on Investment (ROI). *Investopedia*. URL: https://www.investopedia.com/terms/r/returnoninvestment.asp (дата звернення: 07.05.2026).

13. Patterns of Enterprise Application Architecture. *Martin Fowler*. URL: https://martinfowler.com/eaaCatalog (дата звернення: 16.05.2026).

14. Chart.js Documentation. URL: https://www.chartjs.org/docs/latest (дата звернення: 18.05.2026)

15. Background tasks with hosted services in ASP.NET Core. *Microsoft*. URL: https://learn.microsoft.com/aspnet/core/fundamentals/host/hosted-services (дата звернення: 18.05.2026).

ДОДАТОК А

Приклади API-запитів та структур даних системи

А.1 Приклад отримання списку ігрових предметів

У системі реалізовано API для отримання каталогу цифрових активів Counter-Strike 2 із можливістю фільтрації, пошуку та сортування даних.

Запит до сервера:

GET /api/Skin.

На рисунку А.1 наведено параметри запиту для отримання списку ігрових предметів.

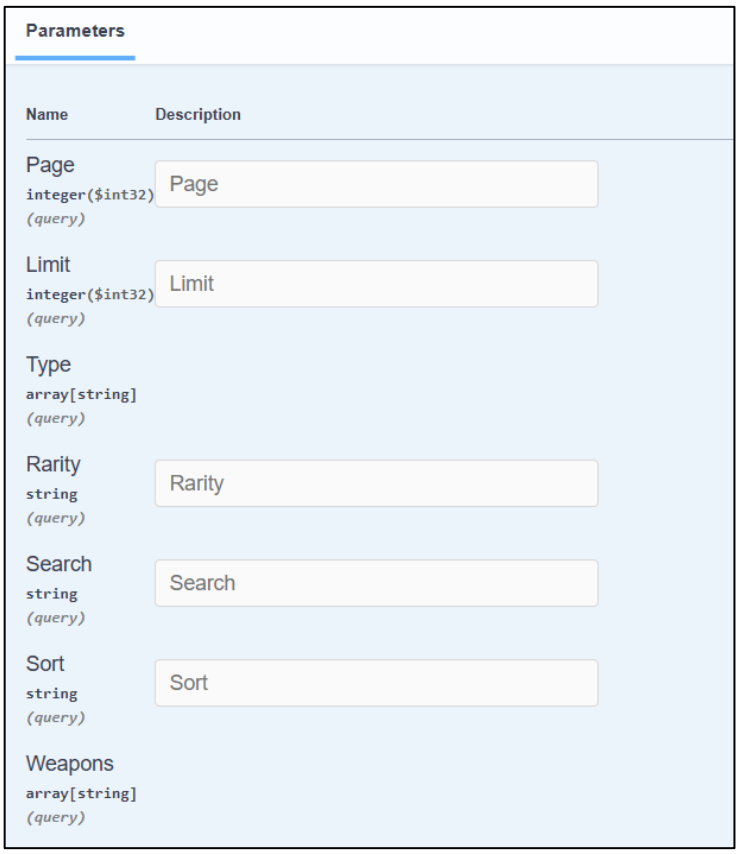


Рисунок А.1 – Параметри запиту для отримання списку ігрових предметів (API /api/Skin)

На рисунку А.2 наведено отримання списку ігрових предметів (API /api/Skin).

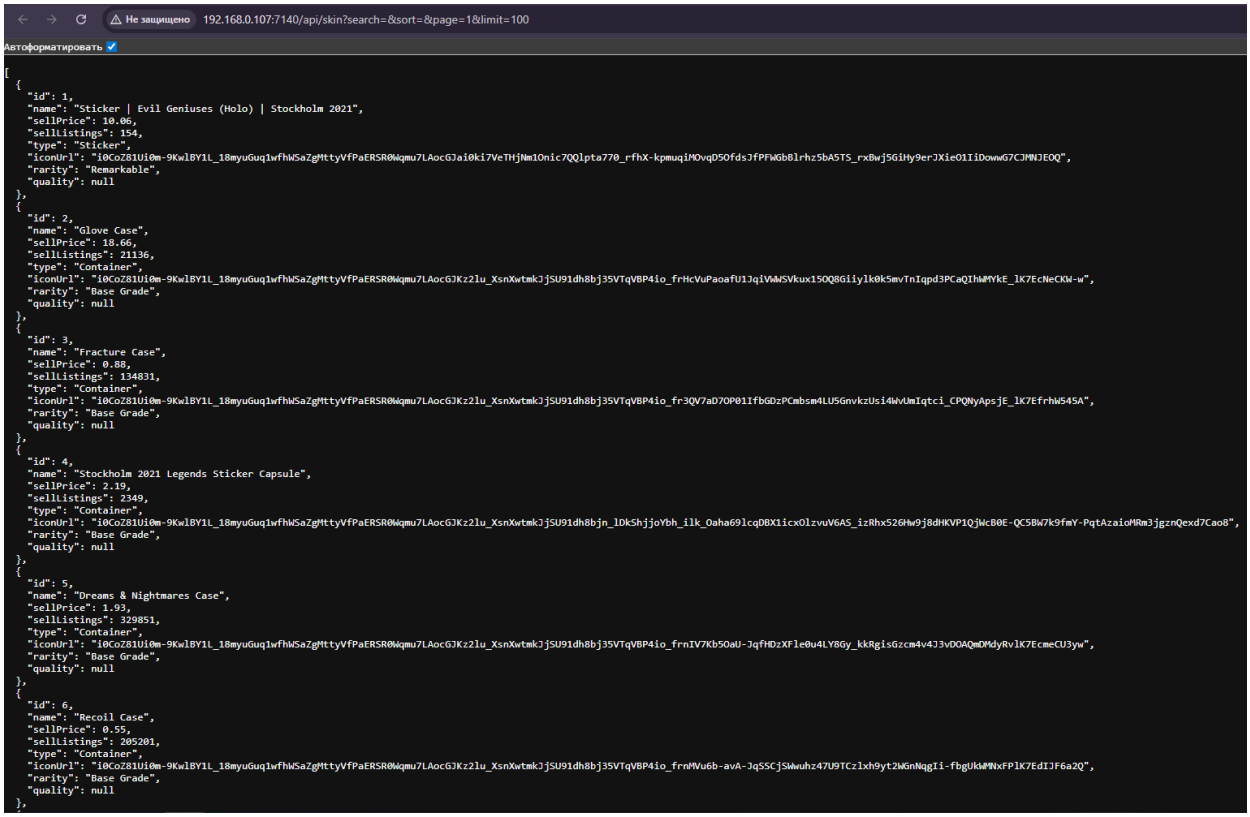


Рисунок А.2 – отримання списку ігрових предметів (API /api/Skin)

А.2 Приклад отримання даних інвестиційного портфеля

У системі реалізовано API для отримання інформації про інвестиційний портфель користувача, включаючи його загальну вартість, прибутковість та рентабельність інвестицій.

Запит до сервера:

GET /api/portfolio/{id}.

На рисунку А.3 наведено приклад отримання даних інвестиційного портфеля.

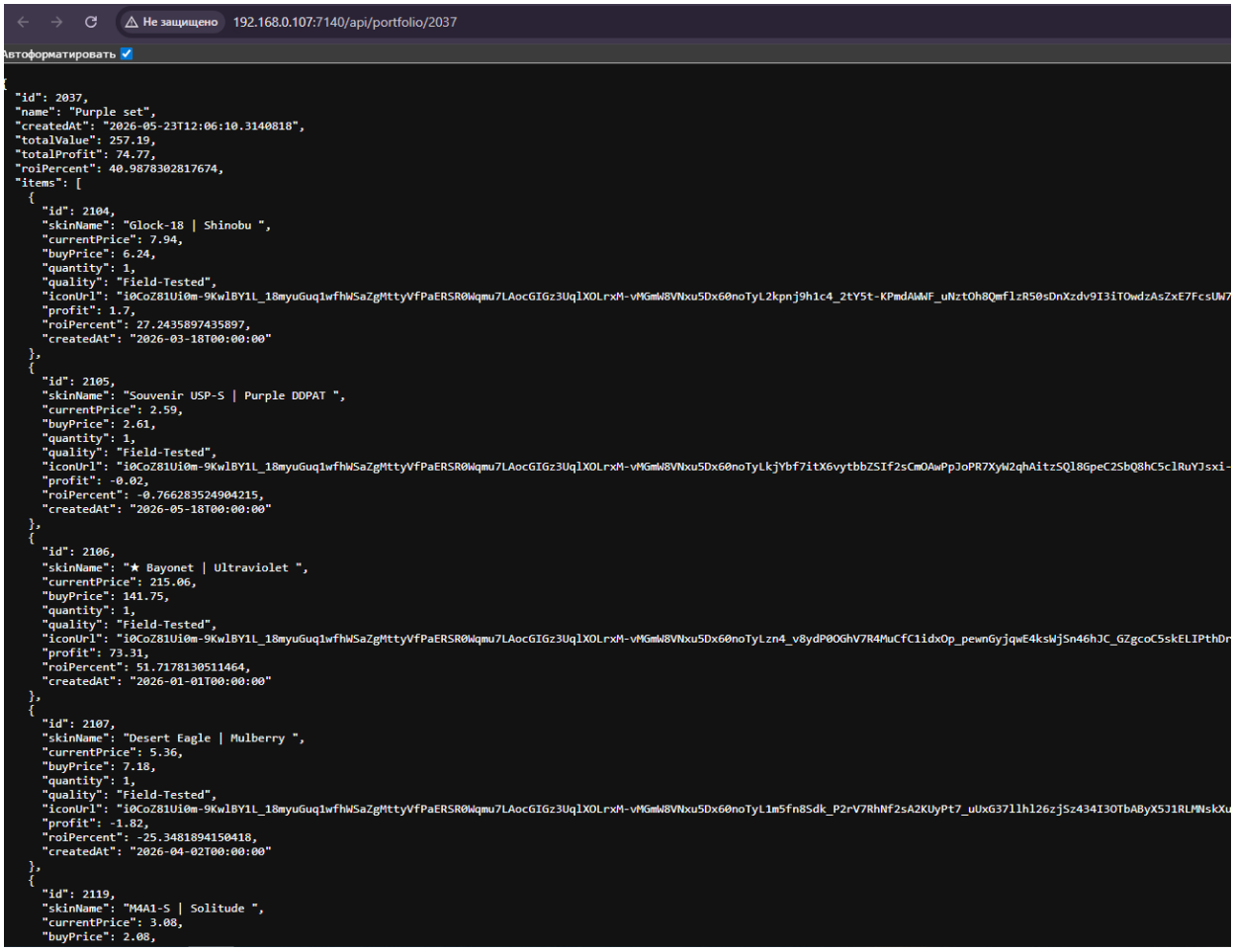


Рисунок А.3 – Отримання даних інвестиційного портфеля

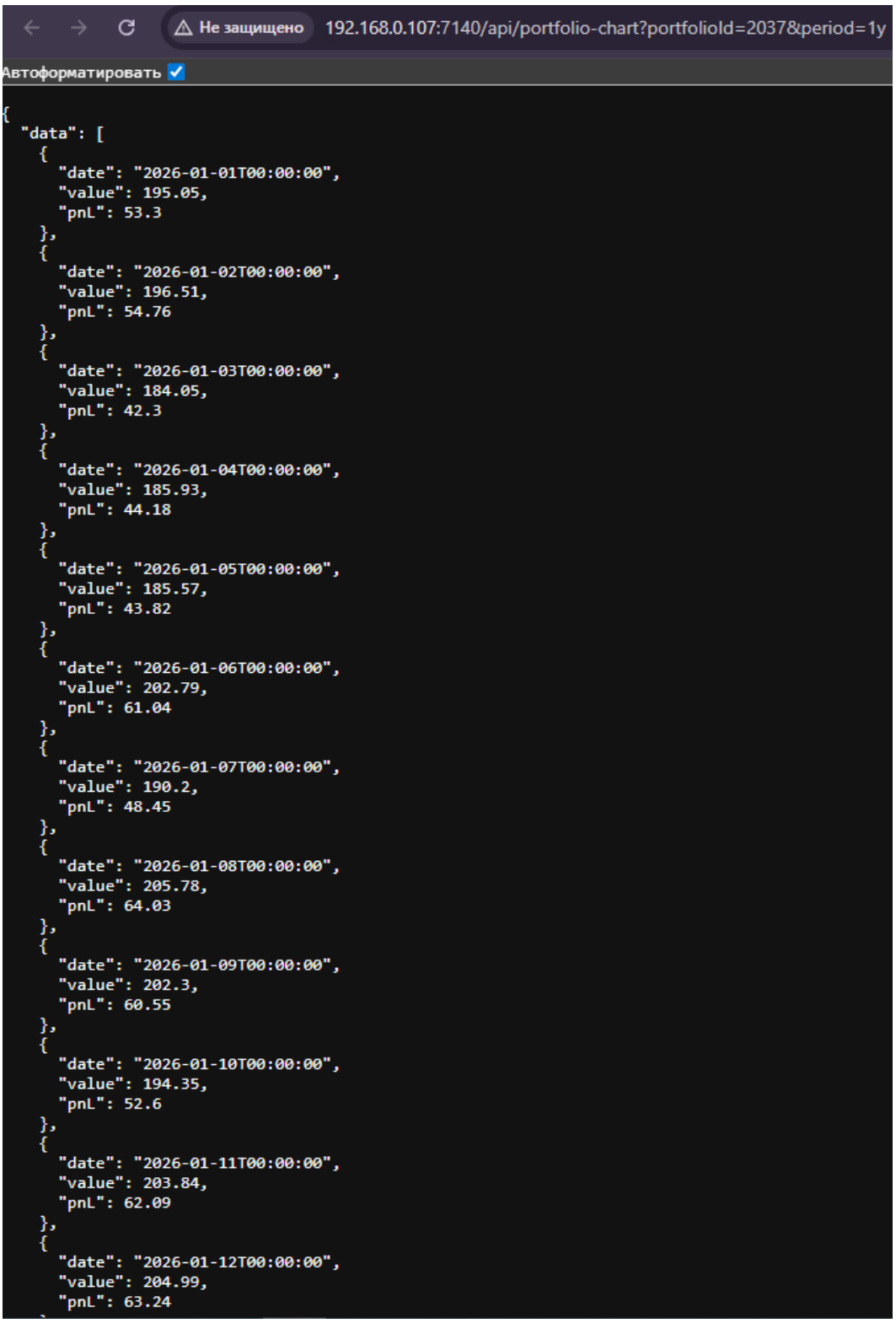
А.3 Приклад даних для побудови графіка динаміки інвестиційного портфеля

У системі аналітики інвестиційних портфелів графік зміни вартості будується на основі історичних розрахованих значень портфеля за певні дати.

Запит до API:

GET /api/portfolio-chart?portfolioId=1&period=7d.

На рисунку А.4 наведено дані, які використовуються для побудови графіка динаміки інвестиційного портфеля.




```
{
  "data": [
    {
      "date": "2026-01-01T00:00:00",
      "value": 195.05,
      "pnL": 53.3
    },
    {
      "date": "2026-01-02T00:00:00",
      "value": 196.51,
      "pnL": 54.76
    },
    {
      "date": "2026-01-03T00:00:00",
      "value": 184.05,
      "pnL": 42.3
    },
    {
      "date": "2026-01-04T00:00:00",
      "value": 185.93,
      "pnL": 44.18
    },
    {
      "date": "2026-01-05T00:00:00",
      "value": 185.57,
      "pnL": 43.82
    },
    {
      "date": "2026-01-06T00:00:00",
      "value": 202.79,
      "pnL": 61.04
    },
    {
      "date": "2026-01-07T00:00:00",
      "value": 190.2,
      "pnL": 48.45
    },
    {
      "date": "2026-01-08T00:00:00",
      "value": 205.78,
      "pnL": 64.03
    },
    {
      "date": "2026-01-09T00:00:00",
      "value": 202.3,
      "pnL": 60.55
    },
    {
      "date": "2026-01-10T00:00:00",
      "value": 194.35,
      "pnL": 52.6
    },
    {
      "date": "2026-01-11T00:00:00",
      "value": 203.84,
      "pnL": 62.09
    },
    {
      "date": "2026-01-12T00:00:00",
      "value": 204.99,
      "pnL": 63.24
```


Рисунок А.4 – Дані які використовуються для побудови графіка динаміки інвестиційного портфеля