

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебплатформа для збору та аналізу ігрової статистики»

Виконав студент групи ПЗ-22-1

БАЛЮК Віталій В'ячеславович

Керівник роботи:

КУШНІР Надія Олександрівна

Рецензент:

ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: БАЛЮК Віталій В'ячеславович

Керівник роботи: КУШНІР Надія Олександрівна

Тема роботи: **«Вебплатформа для збору та аналізу ігрової статистики»**,
затверджена Наказом закладу вищої освіти від «27» лютого 2026 р., №92/с

Вихідні дані для роботи: **об'єктом дослідження є процес розробки функціональної та зручної веборієнтованої платформи для збору та аналізу ігрової статистики, яка дозволить користувачам переглядати власні профілі та історію матчів, аналізувати статистику героїв і предметів за рангами, відстежувати таблиці лідерів за регіонами та стежити за оновленнями гри.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Кушнір Н.О.	02.03.2026р.	02.03.2026р.
2	Кушнір Н.О.	02.03.2026р.	02.03.2026р.
3	Кушнір Н.О.	02.03.2026р.	02.03.2026р.

РЕФЕРАТ

Випускна бакалаврська робота складається з програмного комплексу — веборієнтованої платформи для збору та аналізу ігрової статистики. Пояснювальна записка до бакалаврської роботи містить текстову частину, яка викладена на 82 сторінках друкованого тексту та має 11 сторінок додатків. Список використаних джерел містить 31 найменування і займає 3 сторінки. В роботі наведено 34 рисунки та 16 таблиць. Загальний обсяг роботи — 93 сторінки.

Метою роботи є дослідження та розробка веборієнтованої платформи для збору, агрегації та аналізу ігрової статистики засобами мови програмування TypeScript із застосуванням фреймворків Next.js (React) і Express (Node.js), реляційної СУБД PostgreSQL та сховища даних у пам'яті Redis.

У роботі проаналізовано наявні аналоги програмних продуктів, визначено вимоги користувача та обґрунтовано концептуальну модель системи. Обґрунтовано вибір клієнт-серверної архітектури з шаровою організацією серверної частини та обрано інструментальні засоби реалізації. Спроектовано структуру бази даних і механізм дворівневого кешування даних зовнішніх API. Розроблено програмну реалізацію платформи: автентифікацію користувачів через протокол Steam OpenID 2.0, REST API для агрегації статистики із зовнішніх сервісів, адаптивний інтерфейс користувача з візуалізацією аналітичних даних, а також контейнеризоване розгортання системи засобами Docker.

Ключові слова: ВЕБПЛАТФОРМА, ІГРОВА СТАТИСТИКА, КЛІЄНТ-СЕРВЕРНА АРХИТЕКТУРА, TYPESCRIPT, NEXT.JS, REACT, NODE.JS, POSTGRESQL, REDIS, REST API.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		3

ABSTRACT

This bachelor's thesis consists of a software package — a web-oriented platform for collecting and analyzing game statistics. The explanatory note to the thesis comprises 82 pages of printed text and 11 pages of appendices. The list of references contains 31 entries and spans 3 pages. The thesis includes 34 figures and 16 tables. The total volume of the work is 93 pages.

The aim of the thesis is to research and develop a web-oriented platform for collecting, aggregating, and analyzing game statistics using the TypeScript programming language with the Next.js (React) and Express (Node.js) frameworks, the PostgreSQL relational DBMS, and the Redis in-memory data store.

The work analyzes existing software analogues, identifies user requirements, and substantiates the conceptual model of the system. The choice of a client-server architecture with a layered organization of the server side is justified, and the implementation tools are selected. The database structure and a mechanism for two-level caching of data from external APIs are designed. The software implementation of the platform has been developed: user authentication via the Steam OpenID 2.0 protocol, a REST API for aggregating statistics from external services, an adaptive user interface with visualization of analytical data, and containerized deployment of the system using Docker.

Keywords: WEB PLATFORM, GAME STATISTICS, CLIENT-SERVER ARCHITECTURE, TYPESCRIPT, NEXT.JS, REACT, NODE.JS, POSTGRESQL, REDIS, REST API.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT.....	4
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ВИМОГ ТА ТЕХНОЛОГІЙ СТВОРЕННЯ ВЕБПЛАТФОРМ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту.....	13
1.3. Обґрунтування вибору інструментальних засобів	18
Висновки до першого розділу	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ	24
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	24
2.2. Розробка бази даних системи	29
2.3. Проєктування та реалізація алгоритмів роботи системи.....	34
2.4. Реалізація функціоналу платформи	40
Висновки до другого розділу.....	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ	50
3.1. Порядок встановлення та налаштування параметрів платформи.....	50
3.2. Структура інтерфейсу платформи для збору та аналізу ігрової статистики.....	53
3.3. Тестування платформи	72
Висновки до третього розділу	76
ВИСНОВКИ	78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	81
ДОДАТКИ	84

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

СУБД – система управління базами даних

JWT – токен автентифікації

REST – архітектурний стиль побудови програмних інтерфейсів

ORM – об'єктно-реляційне відображення

ACID – набір властивостей транзакцій бази даних: атомарність, узгодженість, ізолюваність, довговічність

CDN – мережа доставки контенту

CORS – механізм спільного використання ресурсів між різними джерелами

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Сучасна індустрія відеоігор за останні десятиліття трансформувалася з розважальної ніші у глобальний соціокультурний та економічний феномен, обсяг якого зіставний із провідними галузями розважальної індустрії — кіно- та музичною. Особливо помітним є розвиток змагальних онлайн-ігор, які об'єднують мільйони гравців по всьому світу та сформували окремий напрямок — кіберспорт із професійними командами, турнірами та багатомільйонними призовими фондами. Постійне вдосконалення технологій, розширення доступу до мережі Інтернет і популяризація стримінгових платформ сприяють зростанню інтересу як з боку гравців, так і з боку глядачів.

Характерною рисою змагальних ігор є те, що кожен матч породжує великий обсяг структурованих даних: результати команд, показники ефективності окремих гравців, вибір персонажів і предметів, динаміка економіки матчу. Самі по собі ці дані розпорошені та малопридатні для сприйняття — їхня цінність розкривається лише після збору, агрегації та наочного подання. Саме цю задачу вирішують платформи ігрової аналітики, які стали невід'ємним елементом екосистеми кожної успішної змагальної гри: для Dota 2 це Dotabuff і OpenDota, для League of Legends — OP.GG, для Counter-Strike — Leetify. Подібні сервіси щомісяця обслуговують десятки мільйонів користувачів, що свідчить про стійкий попит на ігрову аналітику.

Роль таких платформ в ігровій екосистемі багатогранна. Для пересічного гравця вони виконують функцію персонального аналітика: дозволяють відстежувати власний прогрес, виявляти сильні та слабкі сторони свого стилю гри, порівнювати себе з іншими та приймати обґрунтовані рішення щодо вибору героїв і предметів. Для досвідчених гравців і команд це інструмент підготовки: аналіз статистики суперників, вивчення актуальної мети (найефективніших стратегій поточної версії гри), оцінка балансу персонажів за рангами. Для ширшої спільноти такі сервіси виконують довідкову функцію — агрегують інформацію про оновлення гри, зміни балансу та активність розробників, стаючи єдиною точкою доступу до розпорошених джерел. У

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

такий спосіб аналітичні платформи сприяють підвищенню загального рівня гри спільноти та подовжують життєвий цикл самої гри.

Об'єктом даної роботи обрано гру Deadlock — змагальну онлайн-гру, що активно розвивається. Її аналітична екосистема перебуває на етапі формування: офіційний клієнт гри надає обмежену статистику, а ринок сторонніх інструментів ще не насичений. Це створює практичну потребу в розробці платформи, яка б агрегувала ігрові дані з відкритих API та подавала їх у зручному для аналізу вигляді. Водночас розробка такої системи є показовою інженерною задачею: вона вимагає інтеграції з кількома зовнішніми API, обробки великих обсягів даних, продуманої стратегії кешування та побудови відгукливого інтерфейсу, що робить її актуальною темою в галузі інформаційних технологій.

Метою роботи є розробка функціональної та зручної веборієнтованої платформи для збору та аналізу ігрової статистики, яка дозволить користувачам переглядати власні профілі та історію матчів, аналізувати статистику героїв і предметів за рангами, відстежувати таблиці лідерів за регіонами та стежити за оновленнями гри.

З метою досягнення поставленої мети були сформульовані наступні завдання:

1. проаналізувати наявні аналоги платформ ігрової аналітики та оцінити їхній функціонал;
2. визначити вимоги до системи та обрати архітектуру web-орієнтованої платформи;
3. обрати інструментальні засоби розробки;
4. спроектувати структуру бази даних та механізми кешування даних зовнішніх API;
5. розробити платформу із застосуванням сучасних вебтехнологій;
6. провести тестування та налагодження платформи.

Об'єктом дослідження є процес збору, агрегації та аналітичного подання ігрової статистики змагальних онлайн-ігор у браузерному середовищі.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

Предметом дослідження є методи та засоби реалізації веборієнтованої платформи ігрової аналітики на основі клієнт-серверної архітектури. Дослідження включає аналіз підходів до інтеграції із зовнішніми ігровими API, вибір стратегій кешування та агрегації великих обсягів статистичних даних, проєктування архітектури системи з розподілом відповідальності між клієнтською та серверною частинами, а також реалізацію автентифікації користувачів через сторонній ігровий сервіс.

Методи дослідження. В роботі використано методи системного аналізу для дослідження предметної області, виявлення вимог користувачів та порівняння наявних аналогів платформ ігрової аналітики; методи об'єктно-орієнтованого проєктування і сучасного вебпрограмування для побудови шарової архітектури системи, організації кешування даних та написання серверної й клієнтської частин платформи.

Практичне значення одержаних результатів полягає у тому, що розроблена вебплатформа може бути безпосередньо використана гравцями спільноти гри Deadlock для відстеження власного прогресу, аналізу історії матчів та обґрунтованого вибору героїв і предметів на основі статистики за рангами, а також застосовуватися командами й аналітиками при підготовці до змагань для вивчення статистики суперників та актуального балансу гри. Використання платформи дозволяє об'єднати в єдиній точці доступу дані, розпорошені між кількома зовнішніми джерелами (ігрова статистика, таблиці лідерів, оновлення гри, повідомлення розробників), автоматизувати агрегацію великих обсягів матчевих даних та значно скоротити час отримання аналітичної інформації завдяки багаторівневому кешуванню.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				9
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ТА ТЕХНОЛОГІЙ СТВОРЕННЯ ВЕБПЛАТФОРМ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ

1.1. Постановка задачі

Ігрова аналітика — це збір, агрегація та наочне подання даних, які породжує кожен зіграний матч: результати команд, показники окремих гравців, вибір героїв і предметів, динаміка ігрової економіки. Самі по собі ці дані зберігаються на серверах розробника гри у «сирому» вигляді й малопридатні для сприйняття — цінність створює саме їх структурування та візуалізація. Особливість предметної області полягає в тому, що платформа не породжує первинні дані, а будує власний шар обробки над відкритими API: дані потрібно отримати, зіставити між собою (наприклад, пов'язати ігровий обліковий запис зі Steam-профілем), агрегувати та закешувати так, щоб користувач отримував відповідь за частки секунди, а зовнішні сервіси не блокували платформу через перевищення лімітів запитів. Така архітектура «надбудови над чужими даними» накладає два наскрізні обмеження, що визначають усі подальші технічні рішення: по-перше, платформа не контролює доступність і швидкодію першоджерел, а отже має передбачати їх тимчасові відмови та сповільнення; по-друге, обсяг запитів до зовнішніх сервісів обмежений квотами, тому повторне звернення по ті самі дані є неприпустимою розкішшю й має замінюватися зверненням до власного кешу.

Функціонально платформа складається з кількох програмних модулів. Модуль профілів гравців є ядром системи: за Steam-ідентифікатором або посиланням на профіль користувач знаходить гравця і бачить його зведену статистику — рейтинг, співвідношення перемог і поразок, історію матчів з пагінацією, агреговані показники за кожним героєм та персональні метрики ефективності.

Модуль героїв відповідає за каталог персонажів гри. Для кожного героя відображаються глобальний вінрейт і пік-рейт, розподіл ефективності за ранговими групами, чотири сигнатурні здібності з тірами прокачування, а також панелі характеристик у стилі ігрового клієнта — зброя, живучість і дух.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Описи здібностей автоматично перекладаються українською мовою з кешуванням перекладів, що робить платформу доступною для україномовної аудиторії.

Модуль предметів подає каталог ігрових предметів із фільтрацією за категоріями та рівнями, статистикою вінрейту і пік-рейту кожного предмета та деревом компонентів — з яких дешевших предметів збирається дорожчий. Модуль матчів відображає детальну сторінку зіграного матчу: склади обох команд, підсумкові показники кожного з дванадцяти гравців та інтерактивні графіки перебігу гри. Для побудови графіків реалізовано завантаження та розбір файлів метаданих повтору з серверів Valve — стиснутих бінарних файлів, з яких видобуваються часові ряди ігрової економіки та статистики; це найскладніша частина роботи.

Модуль таблиці лідерів показує рейтинги найкращих гравців за п'ятьма регіонами світу. Оскільки зовнішній API не надає прямого зв'язку між записом у рейтингу та обліковим записом гравця, реалізовано двофазний механізм: швидкий знімок таблиці повертається користувачеві одразу, а точне зіставлення тисяч записів зі Steam-профілями виконується у фоновому режимі з розподіленням блокуванням, після чого кеш оновлюється. Модуль новин агрегує офіційні оновлення гри зі Steam та повідомлення розробників з Telegram-каналу, утворюючи єдину стрічку активності.

Модуль користувачів відповідає за автентифікацію через обліковий запис Steam за протоколом OpenID 2.0: платформа не зберігає паролів, а після підтвердження особи видає власну сесію у вигляді JWT-токена в захищеному httpOnly-cookie [20, 21, 30]. Авторизований користувач отримує швидкий доступ до власного профілю зі статистикою. Весь інший функціонал доступний без входу.

Перелічені модулі не є ізольованими, а утворюють зв'язану предметну модель, у якій сутності посилаються одна на одну: запис історії матчів веде на сторінку матчу, склад матчу — на профілі окремих гравців, профіль агрегує статистику за героями, а та, своєю чергою, відсилає до каталогу героїв і

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

предметів. Завдяки цьому користувач рухається платформою за природними асоціаціями, а не повертається щоразу до пошуку, що й перетворює набір окремих звітів на цілісний аналітичний інструмент. Окрему увагу приділено нефункціональним вимогам, які для аналітичної платформи є не менш важливими за функціональні: малому часу відгуку за прогрітого кешу, стійкості до часткової недоступності зовнішніх сервісів, передбачуваних поведінці інтерфейсу під час завантаження та помилок, а також доступності з мобільних пристроїв завдяки адаптивній верстці.

Окремою інженерною задачею є стратегія кешування. Дані мають різну природу: довідники героїв і предметів змінюються лише з оновленнями гри, статистика матчів — щохвилини, а переклади описів не змінюються ніколи. Тому реалізовано диференційоване кешування у сховищі Redis з різними термінами життя для кожного класу даних, а при розгортанні нової версії сервера похідні кеші очищуються автоматично, що гарантує миттєву появу оновлень без ручного втручання [13, 28]. Така диференціація дозволяє одночасно досягти двох суперечливих цілей — свіжості та швидкодії: дані, що швидко застарівають, мають короткий термін життя й оновлюються майже у реальному часі, тоді як незмінні довідники й переклади кешуються надовго, знімаючи з зовнішніх API основну частину навантаження. Важливою архітектурною властивістю є те, що кеш виступає прискорювачем, а не єдиним джерелом істини: система зберігає працездатність і за повної відсутності Redis, деградує лише у швидкості відповіді, оскільки за відсутності кешованого значення запит прозора перенаправляється до першоджерела.

Розробка системи охоплює проєктування адаптивного інтерфейсу в темній ігровій стилістиці, побудову клієнтської частини на Next.js 16 (React 18) з керуванням серверним станом через TanStack Query, серверної частини на Node.js та Express у форматі REST API, бази даних PostgreSQL через ORM Prisma, а також організацію кодової бази у вигляді монорепозіторію зі спільним пакетом TypeScript-типів, що гарантує узгодженість контракту API між клієнтом і сервером на етапі компіляції. Розгортання виконується у

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Docker-контейнерах. Завершальним етапом є тестування: функціональне, перевірка коректності агрегованих даних відносно першоджерел та оцінка швидкодії за холодного й прогрітого кешу.

1.2. Аналіз аналогів програмного продукту

Перш ніж починати розробку власної платформи, варто розглянути наявні на ринку наявні платформи з такою ж тематикою. Переглянувши та проаналізувавши ці проекти можна знайти як переваги, так і недоліки та виходячи з отриманої інформації створити сервіс, що не поступається конкурентам, але не має їх недоліків.

Першим аналогом є сайт tracklock.gg (рис.1.1 та рис.1.2).

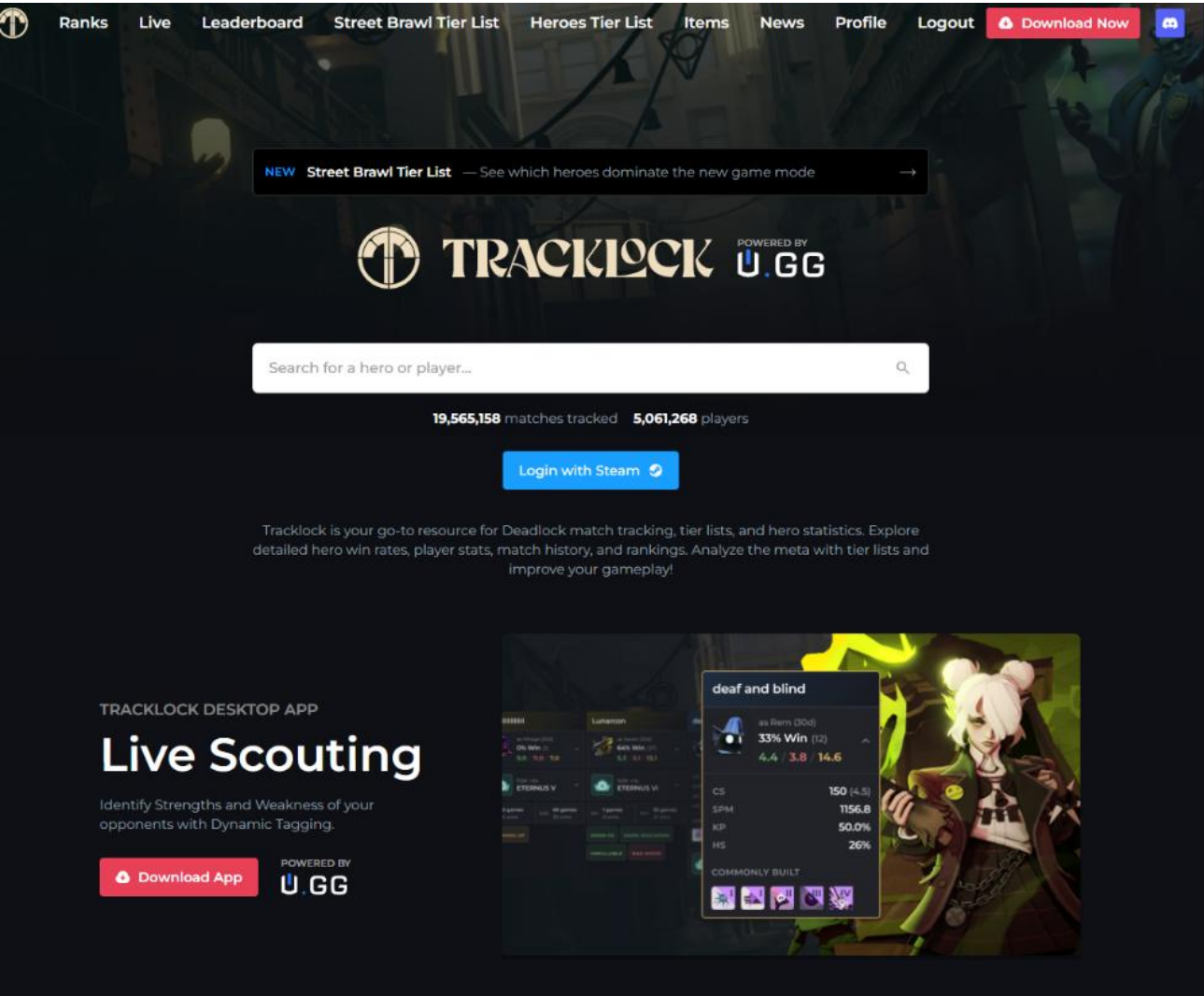


Рис. 1.1. Інтерфейс tracklock.gg



Рис. 1.2. Статистика гравця на сайті tracklock.gg

Tracklock.gg – одна з перших платформ для гри Deadlock, що має простий та зрозумілий інтерфейс. Сервіс працює безоплатно: користувач, ввівши посилання на свій Steam-профіль або ім'я профілю, може отримати загальну статистику свого ігрового облікового запису та історію матчів. Перевагами платформи є швидкість роботи та низький поріг входу — для базового сценарію «знайти себе і подивитися матчі» вона цілком достатня. З очевидних мінусів — статистика матчів досить стисла: відсутні графіки перебігу гри, аналітика героїв обмежена загальними показниками без розподілу за рангами, немає каталогу предметів і агрегації новин гри.

Другим аналогом є Lockblaze (рис.1.3 та рис.1.4).

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

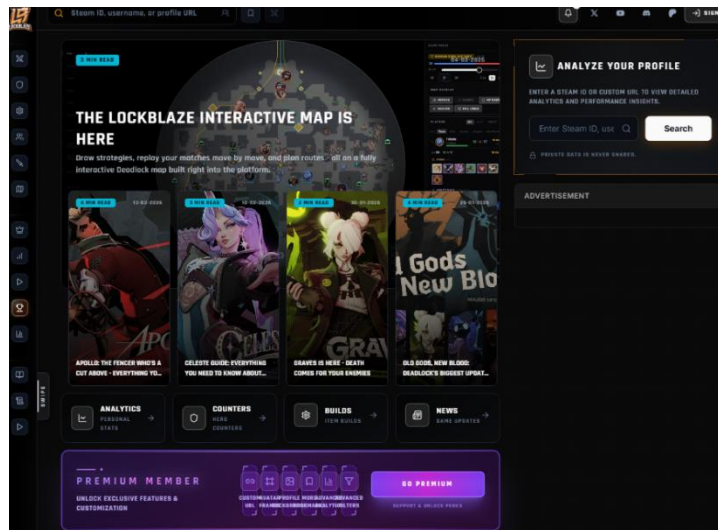


Рис. 1.3. Інтерфейс lockblaze.com

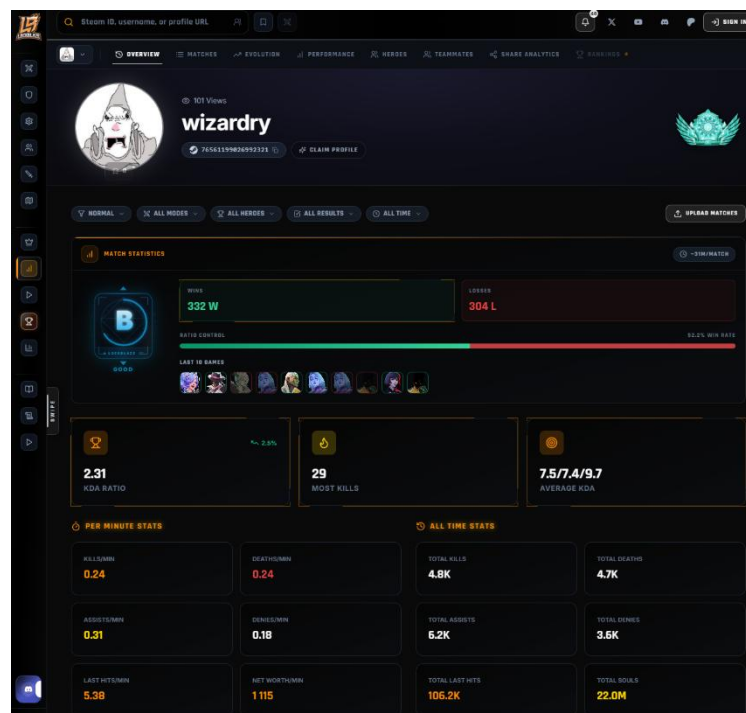


Рис. 1.4. Статистика гравця на сайті lockblaze.com

Платформа має ширший функціонал: синергії персонажів, інтерактивну мапу, розширену статистику гравців. Проте більшість з цих функцій доступна лише за платною підпискою, що суттєво обмежує цінність сервісу для пересічного гравця: безоплатна версія за можливостями наближається до Tracklock.gg. Крім того, інтерфейс перевантажений рекламними блоками та закликами до оформлення підписки, що ускладнює навігацію.

Останній аналог — Statlocker.gg (рис.1.5 та рис.1.6).

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		15



Рис. 1.5. Інтерфейс statlocker.gg.

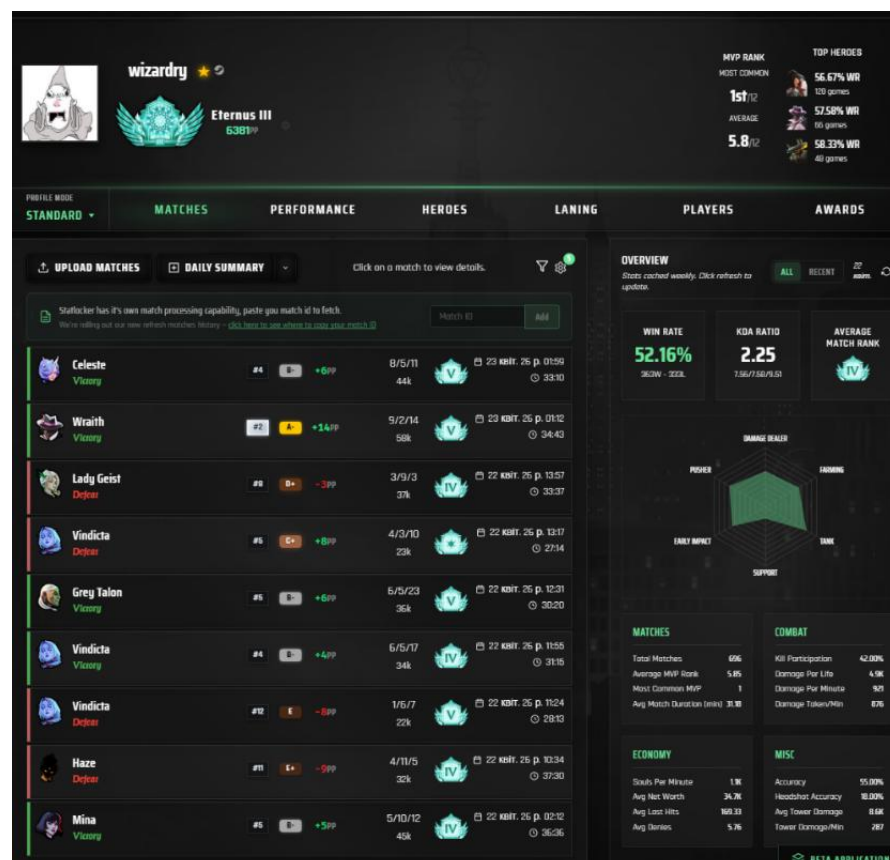


Рис. 1.6. Статистика гравця на сайті statlocker.gg.

Ця платформа має найповніший функціонал серед розглянутих: детальну статистику гравців і матчів, діаграми ефективності гравця порівняно

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

з рештою спільноти, відстеження прогресу відносно минулих результатів. Суттєвим недоліком платформи є погано адаптований до мобільних пристроїв інтерфейс — частина таблиць не вміщується в екран, елементи керування розраховані на курсор

Спільним обмеженням усіх трьох аналогів є відсутність локалізації: інтерфейси доступні лише англійською мовою, а ігрові тексти (описи здібностей, характеристики предметів) подаються в оригінальному вигляді. Для україномовної аудиторії це підвищує поріг входу, особливо для гравців-початківців, які ще не знають англійської термінології гри

Порівняльну характеристику аналогів та розроблюваної платформи наведено в табл. 1.1

Таблиця 1.1

Порівняння платформ

Характеристика	Tracklock	Lockblaze	Statlocker	Розроблювана система
Безоплатний повний функціонал	+	-	+	+
Профіль гравця та історія матчів	+	+	+	+
Порівняння зі спільнотою (перценти́лі)	-	+	+	+
Аналітика героїв за рангами	+	+	+	+
Каталог предметів зі статистикою	+	+	+	+
Графіки перебігу матчу	+	+	+	+
Таблиця лідерів за регіонами	+	+	+	+
Новини гри та активність розробників	-	+	-	+
Українська локалізація	-	-	-	+
Синергії персонажів	-	+	-	-
Інтерактивна мапа	-	+	+	-

За результатами аналізу можна зробити такі висновки. Кожен з аналогів має сильні сторони, але жоден не поєднує повний безоплатний функціонал, та агрегацію всієї інформації про гру в одному місці. Розроблювана платформа

закриває виявлені прогалини: вона безоплатна, поєднує статистику гравців з перцентильним порівнянням, аналітику героїв і предметів за рангами, графіки матчів, регіональні таблиці лідерів та — на відміну від усіх аналогів — єдину стрічку оновлень гри й повідомлень розробників, а також повну українську локалізацію інтерфейсу з автоматичним перекладом ігрових текстів. Водночас слід чесно зазначити обмеження розроблюваної платформи відносно аналогів: у ній відсутні синергії персонажів та інтерактивна мапа (Lockblaze), а обсяг накопиченої історичної статистики поступається сервісам, що працюють від моменту виходу гри. Ці напрямки визначено як перспективи подальшого розвитку системи.

1.3. Обґрунтування вибору інструментальних засобів

Технологічний стек розроблюваної вебплатформи формується з огляду на специфіку поставленої задачі та поєднує мову TypeScript на обох рівнях системи, фронтенд на Next.js 16, бекенд на Node.js з Express, реляційну базу даних PostgreSQL з ORM Prisma та сховище кешу Redis. Вибір кожного компонента підпорядкований двом наскрізним вимогам, що впливають із природи платформи як надбудови над зовнішніми джерелами даних: узгодженості контракту між клієнтом і сервером, що передає одні й ті самі структури даних, та ефективності роботи з операціями введення-виведення, оскільки сервер переважно очікує на відповіді сторонніх сервісів, а не виконує обчислення. Нижче послідовно обґрунтовано вибір кожного з цих компонентів.

Наскрізною мовою розробки обрано TypeScript, що дозволило використати одну мову для клієнтської частини, серверної частини та спільного пакета типів [1, 6]. Це усуває витрати на перемикання між мовними контекстами та, головне, уможливлює ключову архітектурну перевагу проекту — спільний контракт API: структури даних, які сервер формує, а клієнт споживає, описані один раз і перевіряються компілятором з обох боків. Невідповідність полів, яка в динамічно типізованому коді проявилася б збоєм у браузері користувача, тут виявляється ще до запуску застосунку.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Додатковою перевагою є те, що статична типізація виконує роль завжди актуальної документації коду та полегшує безпечний рефакторинг: перейменування поля чи зміна форми відповіді API автоматично підсвічують усі місця, яких ця зміна стосується, що особливо цінно для гри на стадії активного розвитку, де формат зовнішніх даних час від часу змінюється разом із патчами.

Як серверна платформа обрано Node.js з фреймворком Express з огляду на характер навантаження [2, 7, 8]. Сервер платформи майже не виконує обчислень — переважну частину часу він очікує на відповіді зовнішніх API, бази даних та Redis, тобто навантаження є I/O-орієнтованим. Однопоточкова подієва модель Node.js природно відповідає такому профілю: замість того щоб блокувати окремий потік на кожному очікуванні, середовище обслуговує сотні одночасних запитів в межах одного циклу подій, що дає високу пропускну здатність за помірного споживання пам'яті. Express обрано серед фреймворків Node.js завдяки мінімалізму та зрілій екосистемі middleware: захисні заголовки, обмеження частоти запитів, CORS і розбір cookie підключаються готовими модулями кількома рядками конфігурації. Конвеєрна архітектура middleware також природно лягає на потреби платформи — наскрізні задачі на кшталт автентифікації за JWT, валідації вхідних параметрів та централізованого оброблення помилок оформлюються як окремі ланки конвеєра, не змішуючись із бізнес-логікою контролерів.

Для клієнтської частини обрано Next.js замість чистого React через три фактори [3, 9, 10]. По-перше, маршрутизація на основі файлової структури: кожна папка каталогу app автоматично стає сторінкою з власними станами завантаження й помилки, що позбавляє потреби в окремій бібліотеці маршрутизації та її конфігурації. По-друге, вбудована оптимізація зображень, суттєва для платформи, що відображає сотні іконок героїв і предметів зі сторонніх CDN: зображення автоматично масштабуються під розмір контейнера, віддаються у сучасних форматах та підвантажуються відкладено, що зменшує обсяг переданих даних і прискорює першу відмальовку сторінки.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

По-третє, режим збирання standalone, який формує мінімальний самодостатній артефакт для Docker-образу — у підсумковий контейнер потрапляє лише необхідний код без повного дерева залежностей. Керування серверним станом покладено на TanStack Query: для аналітичної платформи, де той самий користувач багаторазово повертається до тих самих даних, клієнтське кешування із фоновим оновленням дає відчутне скорочення часу відгуку без додаткового коду [14]. Бібліотека самостійно бере на себе типові аспекти роботи з віддаленими даними — дедуплікацію одночасних запитів, повторні спроби за тимчасових мережеских помилок, інвалідацію та пагінацію, — що інакше довелося б реалізовувати вручну й узгоджувати між компонентами.

Доступ до реляційної бази даних організовано через ORM Prisma за підходом schema-first: схема описується в декларативному файлі, з якого автоматично генеруються і скрипти міграцій, і типізований клієнт [11]. Завдяки цьому будь-яка зміна структури таблиць проходить через створення нової міграції, відтвореної на іншому середовищі, а згенерований клієнт надає автодоповнення й перевірку типів для кожного запиту — звернення до неіснуючого поля виявляється на етапі компіляції. Типи моделей даних, отримані зі схеми, узгоджуються з наскрізною типізацією проекту, завдяки чому ланцюг «таблиця в базі — об'єкт у коді сервера — відповідь API — модель на клієнті» лишається типобезпечним від краю до краю. У продакшн-розгортанні міграції застосовуються автоматично при старті контейнера сервера, що виключає розбіжність версій коду та схеми. Як СУБД обрано

PostgreSQL — зрілу реляційну систему з підтримкою каскадного видалення залежних записів та індексів за полями фільтрації, які в проекті створено для всіх зовнішніх ключів і пошукових полів [4, 12]. Вибір реляційної моделі зумовлений тим, що дані платформи мають виражену зв'язкову структуру — гравці, матчі, склади команд, герої та предмети поєднані численними відношеннями, — а узгодженість цих зв'язків надійніше забезпечувати засобами цілісності самої СУБД, ніж на рівні застосунку.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

Окремої уваги заслуговує роль сховища Redis, яке зберігає кеш відповідей зовнішніх API з диференційованими термінами життя, описаними в підрозділі 1.3. Окрім кешування, Redis виконує і допоміжну координаційну функцію — слугує сховищем розподілених блокувань, які впорядковують фонове зіставлення записів таблиці лідерів зі Steam-профілями та не дають кільком екземплярам сервера одночасно виконувати одну й ту саму важку операцію. Вибір саме Redis зумовлений тим, що зберігання даних у оперативній пам'яті забезпечує час доступу на рівні часток мілісекунди, а вбудована підтримка терміну життя ключів дозволяє реалізувати диференційоване кешування без додаткового коду для очищення застарілих значень [13, 27].

Для валідації даних застосовано бібліотеку Zod [25]: схеми змінних середовища перевіряються при старті сервера за принципом fail-fast — за відсутності обов'язкового секрету застосунок завершується з явним повідомленням замість прихованих збоїв під час роботи; ті ж засоби використано для валідації параметрів запитів у контролерах. Оскільки зовнішні API є джерелом даних, що не контролюється платформою, перевірка форми відповідей на межі системи захищає від «отруєння» внутрішніх кешів некоректними чи неповними даними. Важливою властивістю Zod є виведення статичних типів безпосередньо зі схем валідації, завдяки чому одна декларація слугує одночасно і перевіркою під час виконання, і джерелом типів під час компіляції. Логування реалізовано структурованим логером Pino, який виводить записи у форматі JSON [23] з рівнями важливості, що уможливлює фільтрацію та аналіз журналів контейнерів у продакшн-середовищі.

Розгортання системи виконується засобами Docker: для кожного артефакту описано багатоетапний образ, де компіляція TypeScript відбувається в проміжному шарі, а до підсумкового образу потрапляє лише скомпільований код [17, 28]. Такий поділ на етапи зменшує розмір кінцевих образів та звужує поверхню атаки, оскільки інструменти збирання й девелоперські залежності до робочого контейнера не потрапляють.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Оркестрація сервісів — сервера, клієнта, бази даних і Redis — описана декларативно засобами Docker Compose, що дозволяє відтворити повне середовище платформи однією командою як на машині розробника, так і на продакшн-сервері, усуваючи розбіжності між середовищами.

Вимоги до клієнтського пристрою мінімальні: платформа повністю виконується у вікні браузера, не потребує встановлення стороннього програмного забезпечення та підтримки апаратного прискорення графіки — достатньо будь-якого сучасного браузера. Адаптивна верстка забезпечує коректну роботу на екранах від мобільних телефонів до широкоформатних моніторів. Серверна частина потребує двох процесорних ядер, чотирьох гігабайтів оперативної пам'яті та десяти гігабайтів дискового простору, а на рівні програмних залежностей — середовища виконання Node.js 20, СУБД PostgreSQL 16 та сховища Redis 7; за використання Docker усі залежності постачаються готовими образами.

Висновки до першого розділу

В першому розділі кваліфікаційної роботи проведено аналіз предметної області та прийнято ключові проектні рішення щодо розроблюваної вебплатформи збору та аналізу ігрової статистики.

Сформульовано постановку задачі: створення безоплатної вебплатформи для гри Deadlock, яка агрегує дані з відкритих API та подає їх у зручному для аналізу вигляді. Визначено функціональний склад системи з семи програмних модулів — профілів гравців, героїв, предметів, матчів, таблиці лідерів, новин гри та автентифікації користувачів. Встановлено, що специфіка предметної області полягає в побудові шару обробки над зовнішніми джерелами даних: платформа не породжує первинні дані, а отримує, зіставляє, агрегує та кешує їх.

Проведено аналіз трьох найпопулярніших аналогів — Tracklock.gg, Lockblaze.com та Statlocker.gg. Виявлено, що жоден з них не поєднує повний безоплатний функціонал та якісну мобільну адаптацію, а спільним обмеженням усіх аналогів є відсутність локалізації. Це визначило нішу

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

розроблюваної платформи: безоплатний повнофункціональний сервіс з українською локалізацією інтерфейсу та ігрових текстів, єдиною стрічкою оновлень гри й адаптивним дизайном.

Обґрунтовано вибір клієнт-серверної архітектури з організацією кодової бази у вигляді монорепозиторію з трьох пакетів та шаровою побудовою серверної частини. Центральним архітектурним рішенням визначено дворівневу систему кешування, що зумовлено лімітами зовнішніх API на частоту запитів. Для автентифікації обрано протокол Steam OpenID 2.0 із сесією на основі JWT-токена в httpOnly-cookie, що звільняє платформу від зберігання паролів.

Таким чином, у результаті виконання першого розділу визначено вимоги до системи, її місце серед наявних аналогів, архітектуру та повний технологічний стек, що створює підґрунтя для проектування структури бази даних і програмної реалізації платформи, розглянутих у наступному розділі.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Перш ніж переходити безпосередньо до реалізації, треба зафіксувати, що саме платформа має робити і хто з нею взаємодіятиме. На цьому етапі набагато дешевше виявити помилку в логіці чи прогалину у вимогах, ніж потім переписувати готові модулі. Тому в цьому параграфі зведено групи користувачів, перелічено функціональні й нефункціональні вимоги, описано ключові варіанти використання та наведено об'єктну модель, на якій базується подальша робота.

Платформа працює у браузері й не вимагає від користувача жодного встановлення. На відміну від систем із розгалуженою рольовою моделлю, тут свідомо обрано мінімальну схему з двох станів — гість та авторизований користувач. Це прямий наслідок призначення системи: аналітична платформа цінна саме відкритістю даних, тому штучне обмеження функціоналу за реєстраційним бар'єром лише відлякувало б аудиторію.

Гість має доступ до всього аналітичного функціоналу: пошуку гравців, перегляду профілів з історією матчів, каталогів героїв і предметів зі статистикою, детальних сторінок матчів з графіками, таблиць лідерів п'яти регіонів, стрічки оновлень гри та повідомлень розробників. Авторизований користувач — це гість, який підтвердив володіння обліковим записом Steam через протокол OpenID 2.0. Авторизація не відкриває прихованого функціоналу, а персоналізує наявний: користувач отримує постійний доступ до власного профілю з навігаційної панелі та особистий кабінет. Реєстрація як окрема дія відсутня — обліковий запис створюється автоматично при першому вході, причому платформа взагалі не зберігає паролів.

Функціональні вимоги до розроблюваної системи визначають спектр її можливостей. Система повинна забезпечувати пошук гравця за кількома форматами ідентифікації; відображення профілю гравця зі зведеною

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

статистикою, оцінкою рангу, пагінованою історією матчів, агрегованими показниками за героями та персональними метриками ефективності з перцентильним порівнянням проти спільноти; ручне оновлення профілю за запитом користувача із захистом від зловживань; каталог героїв з глобальним вінрейтом і пік-рейтом та детальні сторінки героїв із здібностями, панелями характеристик і розподілом ефективності за рангами; каталог предметів з фільтрацією та статистикою використання; детальні сторінки матчів зі складами команд та інтерактивними графіками перебігу гри, побудованими з метаданих повтору; таблиці лідерів п'яти регіонів з коректним відображенням однакових позицій; агрегацію офіційних оновлень гри зі Steam; автентифікацію через Steam без зберігання паролів; автоматичний переклад ігрових текстів українською мовою.

Зовнішній інтерфейс системи оформлено як REST API з ендпоінтами, згрупованими за доменами: автентифікація, гравці, таблиця лідерів, герої, предмети, матчі, патчі, активність розробників. Усі маршрути зведені в одному файлі, що дозволяє побачити повну поверхню API з першого погляду.

Узагальнена діаграма варіантів використання наведена на рис. 2.1. Нижче розглянуто ключові прецеденти.

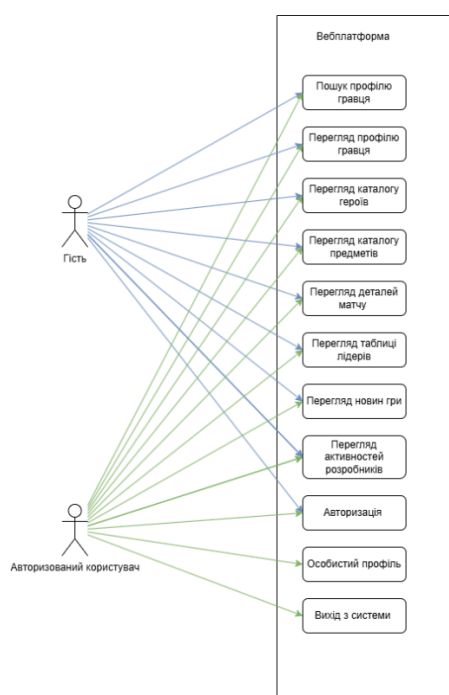


Рис. 2.1. Діаграма варіантів використання

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

Прецедент «Пошук профілю гравця». Основний актор — гість та авторизований користувач. Користувач вводить у пошуковий рядок SteamID64; система нормалізує введення і перенаправляє на сторінку профілю.

Прецедент «Перегляд профілю гравця». Сервер будує «знімок» профілю — єдиний кешований об'єкт, що містить дані гравця, статистику за героями, до 200 останніх матчів та метрики ефективності. Знімок збирається паралельними запитами до Deadlock API і Steam Web API [19, 31], кешується на 7 діб і вважається свіжим протягом доби. Оцінка рангу обчислюється за останніми 50 матчами. Захист від так званого ефекту «лавини» забезпечує розподілене блокування у Redis [26].

Прецедент «Авторизація через Steam». Користувач натискає «Увійти через Steam» і перенаправляється на сторінку steamcommunity.com, де вводить свої облікові дані безпосередньо у Steam — платформа їх не бачить. Steam повертає браузер на ендпоінт зворотного виклику з криптографічно підписаним твердженням, яке сервер верифікує у stateless-режимі. З підтвердженого ідентифікатора витягується SteamID64, профіль збагачується іменем і аватаром, запис користувача створюється або оновлюється у таблиці User, після чого видається JWT-токен з терміном дії 7 діб у httpOnly-cookie.

Прецедент «Перегляд таблиці лідерів». Користувач обирає один з п'яти регіонів; сервер віддає сторінку з кешованого знімка. Особливість прецеденту: зовнішній API повертає для кожного запису лише ігрове ім'я та список можливих ідентифікаторів облікових записів, тому точне зіставлення виконується фоновим скоринговим алгоритмом (параграф 2.3) і не блокує відповідь.

Прецедент «Перегляд деталей героя». Сервер паралельно завантажує метадані героя, аналітику, ранги та всю ігрову інформацію героя, включаючи здібності та характеристики. Кожне з паралельних джерел має власний обробник збою — недоступність аналітики не блокує відображення решти сторінки.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Прецедент «Перегляд деталей предмету». Сервер паралельно завантажує метадані предметів, після чого збирає детальну сторінку. Кожне з паралельних джерел має власний обробник збою — недоступність аналітики не блокує відображення решти сторінки.

Прецедент «Перегляд деталей матчу». Сервер отримує метадані матчу та формує зведення обох команд з показниками дванадцяти гравців. Для графіків перебігу гри окремим запитом отримуються параметри, з яких складається URL файлу метаданих повтору на серверах Valve; файл завантажується, розпаковується з формату bzip2 і розбирається як protobuf-повідомлення [24] у часові ряди статистики кожного гравця.

Прецедент «Перегляд новин гри». Джерелом даних є офіційний Steam News API, з якого відбираються лише офіційні анонси розробника. Кожне оновлення автоматично класифікується за назвою на дрібне, регулярне або мажорне — класифікація використовується для візуального акцентування значних патчів. Steam віддає текст у власній bbcode-розмітці, тому сервер перетворює її на безпечний HTML за принципом «спочатку екранувати все, потім відновити лише відомі теги» — на виході можливі виключно дозволені елементи розмітки, що виключає впровадження стороннього коду через вміст новини.

Прецедент «Перегляд активностей розробників». Користувач переглядає стрічку повідомлень розробників Deadlock, які збираються із Telegram-каналу спільноти, що агрегує їхню активність з офіційного форуму. Таке рішення використовується через відсутність на поточному етапі безперервного моніторингу форуму вебплатформою. Сервер автоматично зчитує нові повідомлення з каналу, зберігає їх у власній базі даних та відображає користувачам. Під час першого запуску завантажується вся доступна історія повідомлень, а надалі додаються лише нові записи. Процес синхронізації працює у фоновому режимі та не впливає на доступність уже збережених даних у разі виникнення помилок.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

Аналіз функціональних вимог дозволив виокремити сім доменних сутностей, реалізованих як моделі ORM Prisma; їхні поля та зв'язки показано на ER-діаграмі (рис. 2.2), а детальний опис наведено у параграфі 2.2. Поведінку системи реалізують сервіси та бібліотечні модулі — основні з них зведені у табл. 2.1.

Таблиця 2.1

Доменні сервіси та бібліотечні модулі серверної частини

Сервіс	Призначення	Ключові методи
playerService	Профілі гравців, історія матчів, метрики	getProfileBySteamId, getMatchHistory, getHeroStats
heroService	Герої: статистика, здібності, характеристики	getAllHeroesWithStats, getHeroDetail
leaderboardService	Сторінки таблиці лідерів	getLeaderboard
devsActivityService	Синхронізація постів розробників	sync, getAll
lib/playerSnapshot	Знімки профілів	getPlayerSnapshotStats, refreshPlayerSnapshot
lib/resolve	Ідентифікатори профілів	resolveAccountIds
lib/deadlockApi	Клієнт Deadlock API	getPlayer, getMatch, getLeaderboard
lib/deadlockReplay	Завантаження й розбір метаданих повторів	parseMatchReplayStats, parseMatchItems
lib/steamNews	Патч-ноути зі Steam News API	fetchSteamPatchNotes
lib/translate	Переклад текстів з кешуванням	translateToUkrainian

Контролери тримають мінімум логіки: типовий метод валідує параметри запиту, делегує виконання сервісу і повертає результат у стандартизованій обгортці; помилки перетворюються на коректні HTTP-відповіді централізованим обробником. Форму даних в обміні описують

TypeScript-інтерфейси спільного пакета, які використовуються одночасно серверним кодом при формуванні відповіді та клієнтським при її

споживанні — зміна структури на одному боці одразу підсвічується компілятором на іншому. Отже, об'єктна модель побудована на чіткому поділі відповідальності: моделі Prisma описують стан, сервіси — поведінку, контролери — транспорт, спільні типи — формат обміну.

2.2. Розробка бази даних системи

Сховищем даних виступає PostgreSQL версії 16 — серед безкоштовних реляційних СУБД він має найкращу комбінацію надійності, продуктивності та функціональних можливостей: ACID-транзакції, масиви як вбудований тип даних, складені та часткові індекси.

Доступ до бази здійснюється через ORM Prisma за підходом schema first: схема описується у декларативному файлі, з якого генеруються одночасно SQL-міграції та типізований клієнт. Історія міграцій зберігається у репозиторії і відстежується системою контролю версій.

Принципова особливість схеми — її компактність відносно обсягу даних, з якими працює платформа. У базі зберігаються лише сутності, що накопичуються самою платформою; велика й мінлива статистика свідомо не дублюються у PostgreSQL — вона отримуються із зовнішніх API та кешується у Redis з диференційованими термінами життя. База даних складається з семи таблиць; опис кожної наведено у табл. 2.2–2.8.

Таблиця 2.2

Опис таблиці Player

Поле	Тип	Призначення
id	String	Первинний ключ (cuid)
steamId	String	Steam-ідентифікатор, унікальний індекс
avatarUrl	String	Посилання на аватар
wins	Int	Кількість перемог
losses	Int	Кількість поразок
updatedAt	DateTime	Час останньої синхронізації (підтримується ORM автоматично)

Записи Player створюються фоновією синхронізацією при перегляді профілю гравця і слугують джерелом збагачення таблиці лідерів — аватарами та показниками перемог. Поле updatedAt підтримується ORM автоматично і відповідає на питання, коли статистику гравця було синхронізовано востаннє; окремого поля для цього не потрібно.

Таблиця 2.3

Опис таблиці User

Поле	Тип	Призначення
id	String	Первинний ключ (uuid)
steamId	String	Steam-ідентифікатор, унікальний індекс
displayName	String	Відображуване ім'я
avatarUrl	String	Посилання на аватар

Таблиці Player і User розділені свідомо, хоча обидві ідентифікуються Steam-ідентифікатором: Player — об'єкт аналітики (будь-який гравець, чий профіль хтось переглянув), User — суб'єкт автентифікації (той, хто увійшов на платформу). Гравців у системі на порядки більше, ніж користувачів, і їхні життєві цикли не пов'язані: видалення облікового запису користувача не повинно знищувати накопичену аналітику гравця. Платформа свідомо не веде журналу входів: при повторній авторизації оновлюються лише ім'я та аватар зі Steam — система не збирає відомостей, які не використовуються її функціоналом.

Таблиця 2.4

Опис таблиці Match

Поле	Тип	Призначення
id	String	Первинний ключ
matchId	String	Зовнішній ідентифікатор матчу
durationSeconds	Int	Тривалість у секундах
startTime	DateTime	Час початку матчу, індексоване поле

Опис таблиці MatchPlayer

Поле	Тип	Призначення
id	String	Первинний ключ
matchId	String	Зовнішній ключ на матч
playerId	Int	Зовнішній ключ на гравця
heroId	Int	Ідентифікатор героя
team	Int	Команда (0 або 1)
won	Boolean	Результат (перемога/поразка)
kills	Int	Кількість вбивств
deaths	Int	Кількість смертей
assists	Int	Кількість асистів
netWorth	Int	Зароблена вартість
lastHits	Int	Кількість добивань
denies	Int	Кількість заблокованих добивань

На парі полів (matchId, playerId) стоїть складений унікальний індекс — той самий гравець не може двічі значитися в одному матчі. Додаткові індекси за playerId та heroId прискорюють найчастіші вибірки: історію матчів гравця та статистику за героєм.

Опис таблиці PlayerHeroStat

Поле	Тип	Призначення
id	String	Первинний ключ
playerId	String	Зовнішній ключ на гравця
heroId	Int	Ідентифікатор героя
gamesPlayed	Int	Кількість зіграних ігор
wins	Int	Кількість перемог
totalKills	Int	Сумарна кількість вбивств
totalDeaths	Int	Сумарна кількість смертей

totalAssists	Int	Сумарна кількість асистів
totalNetWorth	Int	Сумарна зароблена вартість
updatedAt	DateTime	Дата оновлення

Складений унікальний індекс (playerId, heroId) гарантує один агрегований запис на пару «гравець–герой»; оновлення виконується атомарною операцією upsert.

Таблиця 2.7

Опис таблиці Hero

Поле	Тип	Призначення
id	Int	Ідентифікатор із зовнішнього API
name	String	Ім'я героя
className	String	Внутрішній клас-ідентифікатор, унікальний індекс
iconUrl	String	Посилання на іконку
updatedAt	DateTime	Час останньої синхронізації довідника

Це єдиний довідник, продубльований у базі: ідентифікатори героїв потрібні для зв'язності записів MatchPlayer навіть за недоступності зовнішнього API. Назви героїв у записах матчів не дублюються — вони відновлюються за heroId з цього довідника.

Таблиця 2.8

Опис таблиці DevPost

Поле	Тип	Призначення
id	Int	Первинний ключ (ID поста в Telegram)
url	String	Посилання на пост
html	String	HTML-вміст повідомлення
text	String	Текстовий вміст повідомлення
date	DateTime	Дата і час публікації, індексоване поле

Зберігаються лише текстові пости розробників— ім'я розробника та його дія на форумі. Стрічка не містить медіавкладень, тож поля для зображень чи відео не передбачені.

Зв'язки між таблицями використовують єдиний режим каскадного видалення, і це свідомий вибір, що відповідає природі даних. Видалення гравця каскадно видалає його записи участі в матчах та агреговану статистику за героями; видалення матчу — записи всіх його учасників. На відміну від систем із обліковими записами користувацького контенту, тут немає потреби у м'якому видаленні чи режимі Restrict: усі похідні записи (статистика, участь у матчах) повністю залежні від батьківської сутності й не мають сенсу окремо від неї, тому при її видаленні мають зникати разом з нею.

Окремо від реляційної бази, але в межах загальної інформаційної моделі, існує великий шар довідкових даних, який свідомо не зберігається у PostgreSQL: повні описи предметів, здібностей героїв, рангів та панелі характеристик. Ці дані отримуються із зовнішнього Deadlock API і кешуються у Redis із тривалим терміном життя, оскільки змінюються лише з оновленнями гри. Зберігати їхні копії у власній базі означало б постійно синхронізувати два джерела істини при кожному патчі. На рівні коду на героїв посилаються через числовий heroId без матеріалізації самого довідника у реляційних таблицях — у базі присутній лише мінімальний кеш метаданих героїв (таблиця Hero), потрібний для зв'язності записів матчів навіть за тимчасової недоступності API. Так само канонічні назви та слоги героїв задані статично у спільному пакеті типів, а не в базі.

Узагальнена ER-діаграма бази даних з усіма таблицями, типами зв'язків та ключовими полями наведена на рис. 2.2. На діаграмі чітко прослідковується центральне положення таблиці Player, від якої відходять зовнішні ключі до записів матчів та агрегованої статистики, а також відокремленість таблиці User як суб'єкта автентифікації. Стрілки зв'язків відображають кардинальність і режим каскадування, що дозволяє з одного погляду оцінити цілісність схеми

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

та зрозуміти, які саме сутності будуть зачеплені при видаленні конкретного запису.

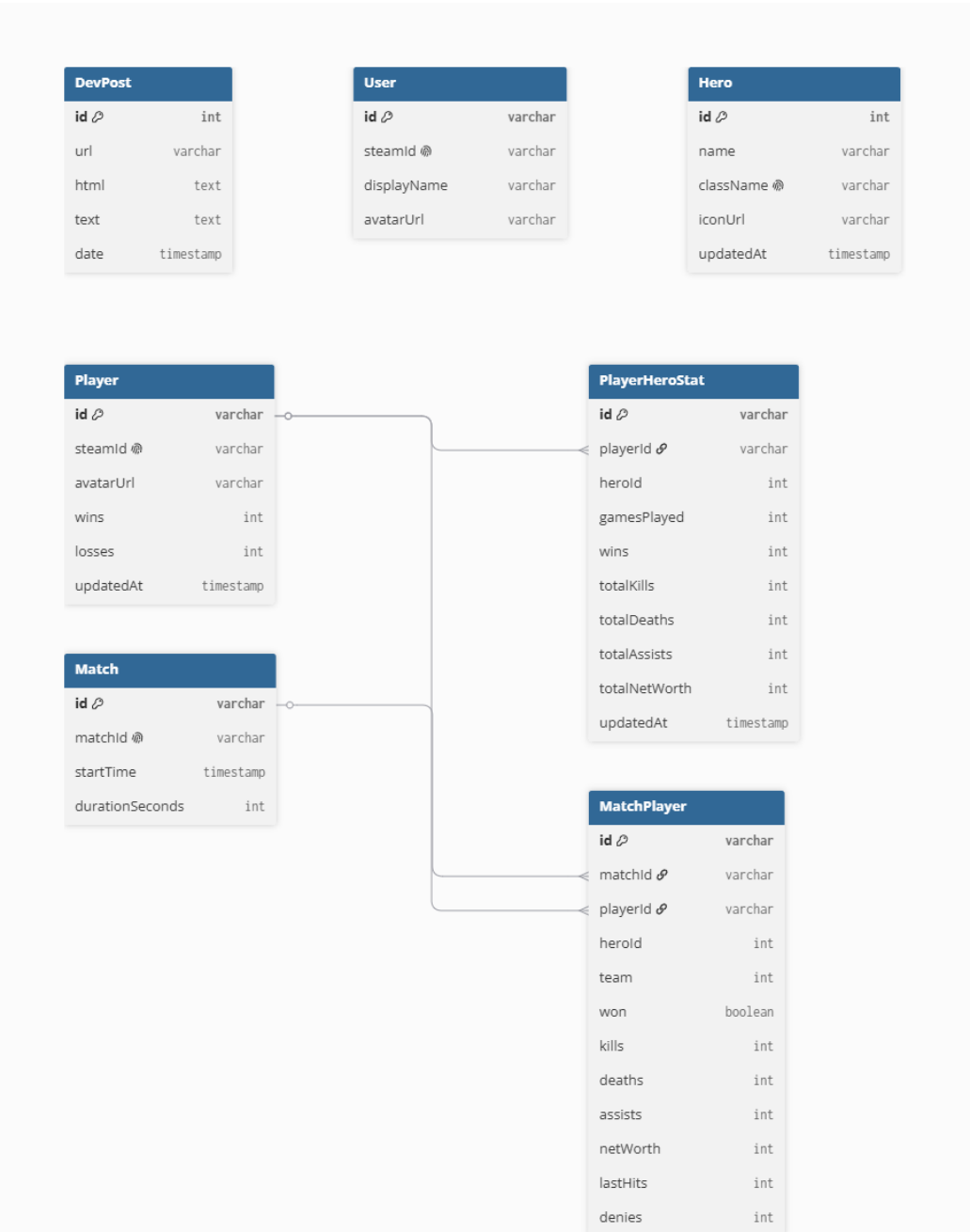


Рис. 2.2. ER-діаграма бази даних

2.3. Проєктування та реалізація алгоритмів роботи системи

За простим інтерфейсом платформи стоїть набір алгоритмів, спроектованих навколо головного обмеження предметної області — лімітів зовнішніх API на частоту запитів. На відміну від систем, що оперують власними даними, ця платформа будує шар обробки над чужими джерелами, тому її алгоритмічне ядро вирішує три наскрізні задачі: швидкий відгук через

багаторівневе кешування, точне зіставлення даних з різних API та стійкість до тимчасової недоступності зовнішніх сервісів. У цьому параграфі описано загальну логіку роботи системи, окремо розглянуто центральний механізм знімків і кешування, а також наведено формули та пояснення до ключових обчислювальних алгоритмів — конвертації ідентифікаторів, розв'язання неоднозначних записів, агрегації статистики, авторизації та безпеки сесій.

Загальний алгоритм роботи з платформою у вигляді UML-діаграми активності винесено у додаток А; нижче коротко описано його основні гілки.

Точкою входу для будь-якого користувача є головна сторінка платформи з пошуковим рядком. Оскільки система свідомо побудована як відкрита аналітична платформа, її розгалуження визначається не роллю, а станом автентифікації — гість чи авторизований користувач. Гість, який не має сесії, отримує повний доступ до аналітичного функціоналу: пошуку гравців, перегляду профілів з історією матчів, каталогів героїв і предметів, детальних сторінок матчів, таблиць лідерів та стрічок новин. Жодна з цих гілок не вимагає входу. Подальший шлях гілкується лише в одній точці — спробі скористатися персоналізованим функціоналом.

Авторизований користувач — це гість, який пройшов вхід через Steam. Авторизація не відкриває прихованих розділів, а додає до інтерфейсу постійне посилання на власний профіль та особистий кабінет зі швидким доступом до персональної статистики. Реєстрація як окрема гілка відсутня: обліковий запис створюється автоматично при першому вході. Вихід із системи очищає сесійний cookie і повертає користувача у стан гостя. Таким чином, діаграма активності платформи має просту топологію з єдиним розгалуженням за наявністю сесії, що відповідає принципу мінімальних бар'єрів для доступу до даних.

Центральним технічним елементом платформи є механізм знімків із багаторівневим кешуванням, що проходить кожен запит до даних гравця. Деталізовану діаграму активності цього механізму наведено у додатку А, а далі описано логіку його станів з опорою на реалізацію у коді.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Знімок профілю — це єдиний кешований об'єкт, що містить дані гравця, статистику за героями, до 200 останніх матчів, метрики ефективності та оцінку рангу. Збирання знімка вимагає кількох паралельних запитів до зовнішніх API, тому результат кешується у Redis на 7 діб і вважається свіжим протягом доби. При зверненні до профілю алгоритм визначає один із трьох станів. Якщо у кеші є свіжий знімок — він повертається миттєво. Якщо знімок є, але застарів — він повертається одразу, а у фоні без блокування відповіді запускається перебудова. Якщо знімка немає — він будується синхронно. Координацію перебудов забезпечує розподілене блокування: за одночасних запитів до того самого профілю перебудову виконує лише один процес, що утримує блокування протягом трьох хвилин, а решта отримують наявну версію. Це усуває ефект «лавини» — ситуацію, коли популярний профіль породжує десятки одночасних звернень до API.

Окремою гілкою цього механізму є захист якості даних. Зовнішній API періодично повертає неповну історію матчів — без ідентифікації героя чи команди. Якщо частка таких записів перевищує половину, новий знімок позначається неповним і не перезаписує наявну якісну версію у кеші, тож тимчасовий збій джерела не псує вже накопичені дані. Ручне оновлення профілю користувачем додатково обмежене кулдауном у 60 секунд на конкретного гравця, що запобігає зловживанню кнопкою.

Із переглядом профілю пов'язана фонові синхронізація у базу даних. Після віддачі знімка користувачеві сервер, не блокуючи відповідь, зберігає гравця у PostgreSQL — рядок Player, записи матчів та агреговану статистику за героями. Частота синхронізації обмежена атомарною операцією Redis на один запис за добу на гравця, тож популярні профілі не породжують зайвих записів. Саме так база поступово накопичує гравців, якими цікавились відвідувачі, і ці записи використовуються для збагачення таблиці лідерів аватарами та показниками перемог.

Базовим обчислювальним алгоритмом є конвертація ідентифікаторів. Steam використовує два формати: 64-бітний SteamID64 та 32-бітний

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

ідентифікатор облікового запису, яким оперує Deadlock API. Перетворення виконується за формулою зсуву на базове значення індивідуальних облікових записів:

$$\text{steamId64} = \text{accountId} + 76561197960265728$$

$$\text{accountId} = \text{steamId64} - 76561197960265728$$

Обчислення виконуються над типом BigInt, оскільки SteamID64 перевищує безпечний діапазон цілих чисел JavaScript ($2^{53} - 1$), а звичайна арифметика дала б похибку округлення. Валідація вхідного ідентифікатора перевіряє довжину в 17 цифр і неменшість за базову константу.

Найскладнішим алгоритмом серверної частини є двофазна побудова знімка таблиці лідерів, що вирішує проблему неоднозначності записів. Зовнішній API повертає для кожного рядка лідерборду — у будь-якому з п'яти регіонів — лише ігрове ім'я та список можливих ідентифікаторів облікових записів, тож кожен такий запис потребує розв'язання. Обсяг роботи різко відрізняється за регіонами: найбільший регіон, Північна Америка, містить близько десяти тисяч записів проти приблизно тисячі в інших, і повне зіставлення всіх його записів зі Steam-профілями триває кілька хвилин — неприйнятно довго для синхронної HTTP-відповіді. Саме цей найгірший випадок зумовив двофазну схему, хоча застосовується вона однаково до всіх регіонів. Перша фаза будує «швидкий» знімок: для кожного запису береться перший кандидат зі списку, а ознака впевненості виставляється лише записам з єдиним кандидатом. Такий знімок повертається користувачеві за секунди. Друга фаза запускається у фоні й виконує повне розв'язання, після чого замінює швидкий знімок у кеші. Координацію фаз забезпечує розподілене блокування з терміном дії 30 хвилин, що гарантує одночасне виконання повільної фази лише одним процесом.

Розв'язання неоднозначного запису у другій фазі реалізовано як скоринговий алгоритм із трьома незалежними сигналами. Для перших восьми кандидатів кожного запису пакетно завантажуються Steam-профілі (до 100 ідентифікаторів за запит) і статистика героїв (до 40 за запит), причому

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		37

кількість одночасних запитів до зовнішнього API обмежено (не більше п'яти), що убезпечує від таймаутів за масового розв'язання. Кожному кандидату нараховується сумарний бал:

$$\text{score} = \text{smatches} + \text{shero} + \text{sname}$$

Визначальним доданком є кількість зіграних матчів: гравці таблиці лідерів мають їх сотні й тисячі, тож великий показник є найнадійнішою ознакою справжності облікового запису. Бал зростає лінійно з числом матчів і обмежений зверху, аби різниця між дуже активними акаунтами не була надмірною:

$$\text{smatches} = \min(40, \lfloor \text{matches} / 40 \rfloor)$$

Другий доданок винагороджує перетин улюблених героїв запису з героями, у яких кандидат має статистику, по 5 балів за кожного спільного:

$$\text{shero} = 5 \times |\text{H}_{\text{запису}} \cap \text{H}_{\text{кандидата}}|$$

Третій доданок оцінює збіг імен і відіграє роль допоміжного критерію (тай-брейкера) за близьких значень попередніх сигналів: точна рівність Steam-імені з ім'ям у лідерборді дає 5 балів, часткове входження одного рядка в інший — 2 бали, відсутність збігу — 0:

$$\text{sname} = 5 \text{ (точний збіг)} \mid 2 \text{ (часткове входження)} \mid 0$$

Таке співвідношення ваг є принциповим: на ранній версії алгоритму збіг імені домінував над кількістю матчів, через що для популярних псевдонімів обиралися малоактивні акаунти з кількома десятками ігор замість справжнього топ-гравця. Перенесення вирішального значення на кількість матчів усунуло цю ваду. Якщо для кандидата взагалі немає даних статистики, він штрафується на 100 балів, що практично виключає його з розгляду. Кандидат із максимальним балом обирається розв'язком. Якщо найкращий бал не сягає порога впевненості у 10 балів, пошук розширюється до сорока кандидатів і скоринг повторюється на ширшому наборі — це дозволяє знайти справжній обліковий запис навіть тоді, коли він не потрапив до першої вибірки серед сотень однойменних кандидатів. Глибоке розв'язання застосовується до провідної частини таблиці (перші позиції, що переглядаються найчастіше),

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

решта записів подаються за першим кандидатом без позначки впевненості. Підсумкова ознака впевненості визначає, чи буде ім'я гравця у таблиці клікабельним посиланням на його профіль: невпевнено розв'язані записи відображаються як простий текст, аби не вести користувача на чужий профіль.

Окрему частину алгоритмічного забезпечення складають механізми авторизації та безпеки сесій. Авторизація реалізована за протоколом Steam OpenID 2.0 і не передбачає зберігання паролів на платформі. На першому кроці сервер формує URL авторизації та перенаправляє браузер на сторінку steamcommunity.com, де користувач вводить облікові дані безпосередньо у Steam. На другому кроці Steam повертає браузер на ендпоінт зворотного виклику з криптографічно підписаним твердженням. Сервер верифікує це твердження у stateless-режимі — без серверного сховища одноразових значень, що спрощує горизонтальне масштабування. З підтвердженого ідентифікатора регулярним виразом витягується SteamID64, профіль збагачується іменем та аватаром (пріоритетним джерелом є Steam Web API, резервним — Deadlock API), а запис користувача створюється або оновлюється атомарною операцією upsert.

Завершальним кроком авторизації є видача сесії. Сервер підписує JWT-токен алгоритмом HS256 із корисним навантаженням, що містить ідентифікатор користувача, SteamID, ім'я та аватар, і встановлює його у cookie з терміном дії сім діб. Cookie має три захисні прапорці: httpOnly робить токен недоступним для JavaScript сторінки, що унеможливлює його викрадення через XSS-атаку; secure дозволяє передачу лише через HTTPS у продакшн-середовищі; sameSite зі значенням lax захищає від міжсайтової підробки запитів. На відміну від систем із серверним сховищем сесій, тут токен є самодостатнім носієм даних сесії: перевірка автентифікації на кожному захищеному запиті зводиться до криптографічної верифікації підпису й не потребує жодного звернення до бази даних. Це робить сервер повністю stateless і дозволяє запускати кілька його екземплярів без спільного стану.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

Стійкість до збоїв зовнішніх сервісів забезпечує алгоритм повторних спроб, яким обгорнуто кожен запит до Deadlock API і Steam Web API. Відповіді зі статусами 429 (перевищення ліміту запитів) та 503 (тимчасова недоступність) не призводять до помилки одразу: алгоритм робить паузу й повторює запит, щонайбільше двічі. Тривалість паузи береться із заголовка Retry-After, якщо сервер його надав (з обмеженням 5 секунд), інакше зростає лінійно з номером спроби. Тайм-аути та решта помилок одразу перетворюються на типізовану помилку зі статусом і контекстом, яку централізований обробник конвертує у коректну HTTP-відповідь для клієнта. Завдяки цьому короточасні збої зовнішніх API не пробиваються до користувача у вигляді відмов.

Останній алгоритм, що заслуговує окремого опису, — кешований переклад ігрових текстів українською мовою, який забезпечує локалізацію описів здібностей і характеристик. Оскільки ці тексти статичні й повторюються між запитами, для кожного з них обчислюється MD5-хеш, що слугує ключем кешу у Redis:

`key = MD5(text)`

За наявності перекладу під цим ключем він повертається миттєво. Інакше текст надсилається до публічного ендпоінта Google Translate, сегменти відповіді склеюються в єдиний рядок, а результат зберігається у кеші на 30 діб. Використання хешу вмісту як ключа дає важливу властивість: кеш ніколи не повертає застарілий переклад, бо будь-яка зміна оригінального тексту дає інший хеш і, відповідно, новий ключ. Так платформа уникає повторних звернень до сервісу перекладу для одного й того самого тексту, зберігаючи при цьому актуальність даних після оновлень гри.

2.4. Реалізація функціоналу платформи

Цей параграф присвячено інженерній стороні проекту — як архітектура, описана раніше, втілена у коді. Розглянуто структуру проекту, технологічні рішення на бекенді й фронтенді, наведено фрагменти найцікавіших ділянок

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		40

реалізації та подано діаграму компонентів системи. Повні лістинги ключових модулів винесено у Додаток Б.

Кодову базу організовано як монорепозиторій на основі npm workspaces: бекенд, фронтенд і спільний пакет типів лежать в одному репозиторії, але збираються незалежно. Спільний пакет shared містить TypeScript-описи всіх структур даних, якими обмінюються клієнт і сервер. Такий підхід синхронізує контракт між клієнтом і сервером на рівні компілятора: при зміні структури відповіді сервера інтерфейс у спільному пакеті правиться один раз, і компілятор одразу підсвічує всі місця на обох боках, де код перестав відповідати новій формі даних. Це усуває цілий клас помилок інтеграції, типових для проектів, де клієнт і сервер описують дані незалежно.

Серверна частина побудована на Node.js із фреймворком Express за шаровою схемою [5, 29]: маршрути приймають HTTP-запити, контролери розбирають параметри й делегують роботу сервісам, сервіси містять бізнес-логіку, а доступ до бази інкапсульовано у шарі репозиторіїв. Центральною точкою збирання застосунку є файл app.ts, де формується конвеєр обробки запитів. Порядок проміжних обробників (middleware) тут критичний:

```
app.use(helmet()); // захисні HTTP-заголовки
app.use(cors({ origin: config.FRONTEND_URL, credentials: true }));
app.use(cookieParser()); // розбір сесійного cookie
app.use(express.json({ limit: "10mb" }));
app.use(morgan(...)); // логування запитів
app.use(rateLimit({ windowMs: 60_000, max: 100 })); // обмеження частоти
app.use("/api", router); // усі маршрути API
app.use(notFound); // 404
app.use(errorHandler); // централізована обробка помилок
```

Спершу встановлюються захисні заголовки й політика CORS, далі розбираються cookie та тіло запиту, потім спрацьовує глобальний обмежувач частоти, і лише після цього запит потрапляє до маршрутів. Замикає конвеєр централізований обробник помилок: будь-яка типізована помилка, кинута

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

глибше у сервісах, перетворюється тут на коректну HTTP-відповідь зі стандартизованою структурою.

Конфігурація застосунку валідується при старті за схемою. Замість того щоб покладатися на наявність змінних середовища під час роботи, сервер перевіряє їх одразу й «падає» з явним повідомленням, якщо щось не так — за принципом fail-fast:

```
const envSchema = z.object({
  DATABASE_URL: z.string(),
  JWT_SECRET: z.string().min(32),
  REDIS_URL: z.string().default("redis://localhost:6379"),
  ...
});
const parsed = envSchema.safeParse(process.env);
if (!parsed.success) {
  console.error("❌ Invalid environment variables:");
  process.exit(1);
}
```

Зокрема, вимога min(32) до JWT_SECRET гарантує, що сервер не запуститься зі слабким секретом підпису токенів. Така перевірка на старті переводить цілий клас помилок конфігурації з категорії «загадковий збій під час роботи» у категорію «зрозуміла відмова при запуску».

Безпека сесій реалізована без зберігання стану на сервері. При успішній авторизації через Steam контролер підписує JWT-токен і кладе його у захищений cookie:

```
const token = await new SignJWT({
  sub: user.id, steamId: user.steamId,
  displayName: user.displayName, avatarUrl: user.avatarUrl,
})
.setProtectedHeader({ alg: "HS256" })
.setIssuedAt()
.setExpirationTime("7d")
.sign.jwtSecret);
```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		42

```
res.cookie("session", token, {
  httpOnly: true, // недоступний для JavaScript (захист від XSS)
  secure: config.NODE_ENV === "production", // лише HTTPS у продакшні
  sameSite: "lax", // захист від CSRF
  maxAge: 7 * 24 * 60 * 60 * 1000, // 7 діб
});
```

Перевірка сесії на кожному захищеному запиті зводиться до криптографічної верифікації підпису й не звертається до бази даних:

```
async function decodeToken(token: string) {
  try {
    const { payload } = await jwtVerify(token, jwtSecret);
    return { id: payload.sub, steamId: payload.steamId, /* ... */ };
  } catch {
    return null; // підпис недійсний або токен прострочений
  }
}
```

Оскільки токен є самодостатнім носієм даних сесії, сервер залишається повністю stateless: жодне його звернення не потребує спільного стану між запитами, що дозволяє запускати кілька екземплярів за балансувальником без додаткової синхронізації. Реалізовано два варіанти захисту маршрутів — requireAuth і optionalAuth — останній застосовано до публічних ендпоінтів, де авторизація лише персоналізує відповідь.

Найцікавішою обчислювальною ділянкою є скоринговий алгоритм розв'язання неоднозначних записів лідерборду, зосереджений у функції scoreEntry. Він не має побічних ефектів — на вхід приймає кандидата й заздалегідь зібрані дані, на вихід повертає бал, що робить його зручним для тестування. Ключова частина — нарахування балів за трьома сигналами:

```
// Сигнал 1: збіг Steam-імені з іменем у лідерборді
if (name === target) score += 10;
else if (name.includes(target) || target.includes(name)) score += 3;

// Сигнал 2: перетин улюблених героїв запису з героями кандидата
```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

```

for (const heroId of entryTopHeroes)
  if (heroData.allHeroIds.has(heroId)) overlap++;
score += overlap * 5;
// Сигнал 3: кількість зіграних матчів (логарифмічне згладжування)
if (heroData.totalMatches > 0)
  score += Math.min(10, Math.floor(Math.log10(heroData.totalMatches) * 4));

```

Логарифмічне згладжування третього сигналу не дає кількості матчів домінувати над рештою: різниця між гравцем на 100 і 1000 матчів суттєва, а між 10 000 і 100 000 — майже ні, що й відображає логарифм. Кандидат із максимальним балом обирається розв'язком, а якщо найкращий бал не сягає порога впевненості у 10 балів, пошук розширюється з п'яти до двадцяти кандидатів.

Доменна логіка кешування з блокуваннями ілюструє неочевидний інженерний сценарій. Побудова знімка таблиці лідерів використовує атомарну операцію Redis для розподіленого блокування, аби повільну фазу одночасно виконував лише один процес:

```

const lockAcquired = await cacheSetIfNotExists(lockKey, "1", LOCK_TTL_SECONDS);
if (lockAcquired) scheduleFullResolution(region, snapKey, lockKey);

```

Сама операція `cacheSetIfNotExists` спирається на прапорець NX (`set-if-not-exists`), що в Redis виконується атомарно:

```

const result = await redis.set(key, value, { NX: true, EX: ttlSeconds });
return result === "OK"; // true, лише якщо ключ не існував

```

Завдяки цьому навіть за десятків одночасних запитів до того самого регіону повну перебудову розпочне рівно один процес, а решта отримають наявну версію даних — це і розв'язує проблему «лавини» одночасних перебудов.

Стійкість до збоїв зовнішніх API забезпечує обгортка повторних спроб, якою огорнуто кожен запит:

```

if ((status === 429 || status === 503) && attempt < MAX_RETRIES) {
  const retryAfter = err.response?.headers?.["retry-after"];
  const delay = retryAfter
    ? Math.min(parseInt(retryAfter, 10) * 1000, 5_000)

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

```

: 1_000 * (attempt + 1);
await new Promise((r) => setTimeout(r, delay));
continue;
}

```

Перевищення ліміту (429) чи тимчасова недоступність (503) призводять до паузи й повтору, причому тривалість паузи береться із заголовка `Retry-After`, якщо сервер його надав. Короточасні збої так не пробиваються до користувача.

Клієнтська частина реалізована на `Next.js 16` із маршрутизацією за файловою структурою `App Router`: кожна папка в каталозі `app` автоматично стає сторінкою з власними станами завантаження й помилки, без окремої бібліотеки маршрутизації. Керування серверним станом покладено на `TanStack Query` — для аналітичної платформи, де користувач багаторазово повертається до тих самих даних, його клієнтське кешування дає відчутне скорочення часу відгуку. Звернення до API оформлені як хуки з оголошеними ключами кешу:

```

export function usePlayer(steamId: string) {
  return useQuery({
    queryKey: QueryKeys.player(steamId),
    queryFn: () => api.players.get(steamId),
    enabled: Boolean(steamId),
    staleTime: 5 * 60 * 1000,      // 5 хвилин «свіжості»
    retry: (count, err) =>
      err.statusCode === 404 ? false : count < 2, // не повторювати 404
  });
}

```

Оголошення ключів кешу окремими константами (`QueryKeys`) дозволяє точково інвалідувати дані: після ручного оновлення профілю мутація скидає саме ключі цього гравця, і відповідні сторінки перезавантажуються автоматично. Поки дані «свіжі» (`staleTime`), повторні переходи між сторінками обслуговуються з клієнтського кешу без звернень до сервера.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Найскладніша ділянка реалізації — розбір файлів метаданих повтору матчу для побудови графіків. Ці файли хостяться на серверах Valve як protobuf-повідомлення, стиснуті алгоритмом bzip2, тож їх обробка проходить три етапи: завантаження, розпакування бібліотекою seek-bzip і десеріалізація за частковим описом протоколу. Опис охоплює лише ті поля, що реально потрібні платформі — protobuf мовчки пропускає решту:

```
message PlayerStats {
  optional uint32 time_stamp_s = 1;
  optional uint32 net_worth = 2;
  optional uint32 kills = 14;
  optional uint32 deaths = 15;
  optional uint32 assists = 16;
  optional uint32 player_damage = 21;
  optional uint32 player_healing = 25;
  optional uint32 self_healing = 27;
  optional uint32 player_damage_taken = 28;
  repeated GoldSource gold_sources = 40;
}
```

З розібраного повідомлення видобуваються часові ряди кожного гравця — нетворс, вбивства, шкода, лікування за часовими позначками, — які фронтенд відображає інтерактивними графіками на бібліотеці Recharts [16].

Поза основними сервісами реалізовано три допоміжні підсистеми. Переклад ігрових текстів українською виконує модуль translate із кешуванням за хешем вмісту: для кожного тексту обчислюється MD5-хеш, який слугує ключем у Redis, тож повторний переклад того самого тексту безкоштовний і миттєвий, а зміна оригіналу автоматично дає новий ключ. Стрічку активності розробників наповнює модуль telegram, що розбирає вебверсію публічного Telegram-каналу і зберігає лише пости з префіксом «Developer:». Патч-ноути формує модуль steamNews: він тягне офіційні анонси зі Steam News API, класифікує їх за типом і перетворює власну bbcode-розмітку Steam на безпечний HTML за принципом «спочатку екранувати все, потім відновити

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

лише дозволені теги», що виключає впровадження стороннього коду через вміст новини.

Підсумувати викладене допомагає діаграма компонентів системи (рис. 2.3), на якій показано основні логічні блоки платформи та зв'язки між ними.

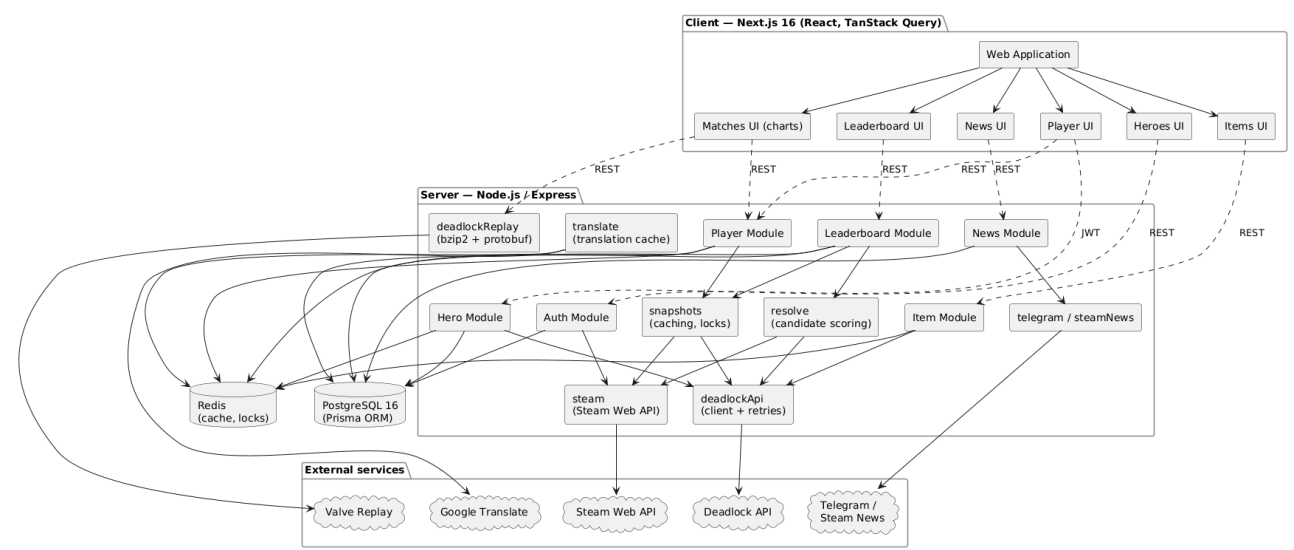


Рис. 2.3. Діаграма компонентів системи

На діаграмі представлено логічні компоненти, згруповані у три рівні. Інфраструктурний рівень формують клієнти зовнішніх API (Deadlock API та Steam Web API з обгорткою повторних спроб), кешуюче сховище Redis (відповіді API, переклади, розподілені блокування) та шар персистенції PostgreSQL через ORM Prisma. На рівні доменної логіки розташовані Player Module (знімки профілів із блокуваннями та фонові синхронізація в БД), Leaderboard Module (двофазне розв'язання неоднозначних записів), Hero Module і Item Module (каталоги зі статистикою та кешуванням довідників), Match Module (зведення матчів і розбір повторів) та News Module (агрегація оновлень гри й активності розробників). Окремо стоїть Auth Module — автентифікація через Steam OpenID із видачею stateless JWT-сесій. Клієнтський рівень — застосунок Next.js, що споживає REST API через хуки TanStack Query.

Модулі взаємодіють «знизу вгору»: доменні модулі звертаються до інфраструктурних (зовнішні API, кеш, база), а самі доступні клієнту через REST-ендпоінти. Така структура дозволяє розвивати систему модульно —

наприклад, додавання нового аналітичного розділу зводиться до нового сервісу, контролера та маршруту, не торкаючись інших підсистем.

Загалом реалізація платформи побудована на простих і перевірених рішеннях замість екзотичних патернів: стандартний Express-конвеєр, типобезпечний доступ до даних через Prisma, кешування на Redis, stateless-автентифікація. Кожне ключове рішення — багаторівневе кешування, розподілені блокування, двофазне розв'язання лідерборду, валідація конфігурації за схемою — має конкретне обґрунтування у вимогах продуктивності та стійкості. Така структура дозволяє новому розробникові зорієнтуватися у проекті швидко, а нова функціональність зазвичай зводиться до однотипних змін у маршруті, контролері та сервісі.

Висновки до другого розділу

У другому розділі виконано проектування та програмну реалізацію вебплатформи збору й аналізу ігрової статистики.

Визначено варіанти використання системи та її об'єктно-орієнтовану структуру. Обґрунтовано мінімальну рольову модель із двох станів — гість і авторизований користувач, — що відповідає призначенню відкритої аналітичної платформи. Описано ключові прецеденти (пошук і перегляд профілю, ручне оновлення, авторизація через Steam, перегляд героїв, матчів, таблиці лідерів і новин) та подано серверну архітектуру у вигляді чотирьох шарів — маршрути, контролери, сервіси й репозиторії — з інкапсуляцією доступу до бази у шарі репозиторіїв.

Спроектовано структуру бази даних із семи таблиць на основі PostgreSQL та ORM Prisma. Схему свідомо побудовано компактною за принципом мінімізації даних: у реляційному сховищі зберігаються лише сутності, що накопичуються самою платформою, тоді як великі довідники та мінливі дані зовнішніх API винесено у кешуюче сховище Redis. Описано обмеження, індекси та єдиний режим каскадного видалення, кожен із яких має конкретне обґрунтування у логіці роботи системи.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

Розроблено алгоритмічне ядро системи, спроектоване навколо головного обмеження предметної області — лімітів зовнішніх API. Наведено формули та реалізацію ключових алгоритмів: конвертації ідентифікаторів Steam, кодування рангів, побудови знімків профілів із розподіленими блокуваннями, двофазного розв'язання неоднозначних записів таблиці лідерів зі скоринговим зіставленням за трьома сигналами, агрегації статистики, stateless-автентифікації на основі JWT та кешованого перекладу ігрових текстів.

Описано програмну реалізацію платформи: організацію кодової бази як монорепозиторію зі спільним пакетом типів, що забезпечує типобезпечний контракт API між клієнтом і сервером; конвеєр обробки запитів Express із захисними механізмами; багаторівневе кешування на Redis із розподіленими блокуваннями; розбір стиснутих файлів повторів матчів. Наведено фрагменти найважливіших ділянок коду, побудовано діаграму компонентів системи та діаграми активності основних процесів.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ЗБОРУ ТА АНАЛІЗУ ІГРОВОЇ СТАТИСТИКИ

3.1. Порядок встановлення та налаштування параметрів платформи

Вебплатформа спроектована як набір незалежних компонентів, що взаємодіють по мережі, тому її розгортання зводиться не до встановлення єдиного застосунку, а до підняття кількох служб і налаштування зв'язків між ними. Архітектуру системи та розподіл програмного забезпечення на обчислювальних вузлах ілюструє діаграма розгортання (рис. 3.1).

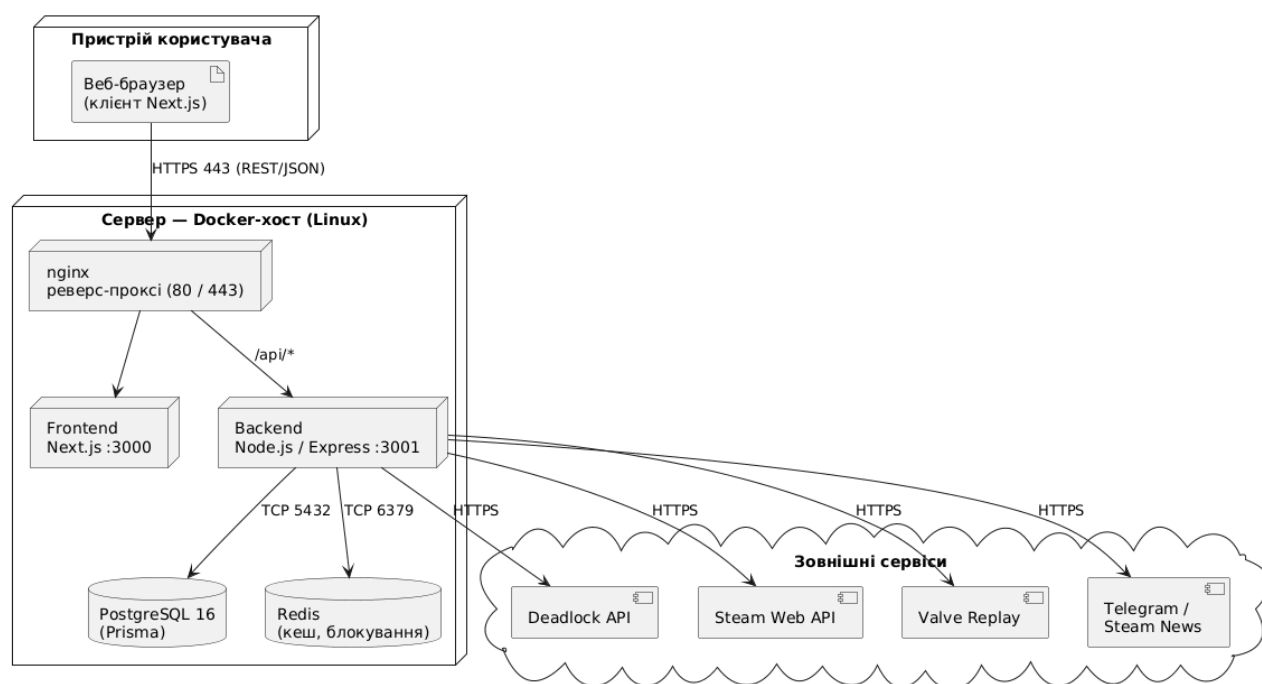


Рис. 3.1. Діаграма розгортання вебплатформи

Архітектура розгортання охоплює три рівні вузлів. На пристрої користувача — персональному комп'ютері або смартфоні — працює веббраузер, у якому виконується клієнтський застосунок, реалізований засобами Next.js. Це єдина точка взаємодії відвідувача із системою, що не вимагає встановлення жодного додаткового програмного забезпечення. Центральним вузлом є сервер під керуванням Docker, на якому в ізольованих контейнерах одночасно працюють п'ять служб: реверс-проксі nginx, що приймає зовнішні запити на портах 80 і 443 [18]; контейнер клієнтської

частини Next.js; контейнер серверної частини Node.js; СУБД PostgreSQL зі сталим томом для зберігання даних; сховище Redis для кешування і розподілених блокувань. Третій рівень утворюють зовнішні сервіси, до яких звертається серверна частина: Deadlock API (ігрова статистика), Steam Web API (профілі гравців і автентифікація OpenID), сервери повторів Valve, Telegram-канал активності розробників та сервіс перекладу.

Взаємодія між вузлами відбувається за чітко визначеною схемою. Браузер обмінюється даними з реверс-проксі nginx по протоколу HTTPS, передаючи запити та отримуючи відповіді у форматі JSON за принципами REST [22]. Реверс-проксі маршрутизує запити всередині сервера: звернення до статичних сторінок направляються контейнеру клієнтської частини, а запити за шляхом /api/* — контейнеру серверної частини. Серверна частина, своєю чергою, звертається до PostgreSQL по TCP на порту 5432 для роботи з накопиченими даними, до Redis на порту 6379 для читання й запису кешу, а до зовнішніх ігрових сервісів — по захищеному HTTPS-з'єднанню. Такий розподіл функцій ізолює служби одну від одної: кожен контейнер можна оновлювати, перезапускати чи масштабувати окремо, не зачіпаючи решти.

Розгортання системи виконується у двох режимах. У режимі розробки в контейнерах піднімаються лише інфраструктурні служби — PostgreSQL і Redis, — а код клієнтської та серверної частин виконується нативно засобами середовища Node.js.

У продакшн-режимі весь стек, включно з клієнтською та серверною частинами й реверс-проксі, піднімається в контейнерах єдиною командою з окремого файлу docker-compose.prod.yml. Образи зібрано за принципом багатоетапної збірки: компіляція TypeScript відбувається у проміжному шарі, а до підсумкового образу потрапляє лише скомпільований код, що зменшує розмір образів і поверхню атаки. Міграції бази даних у цьому режимі застосовуються автоматично при старті контейнера серверної частини, тож окремий крок оновлення схеми не потрібен.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Параметри системи задаються виключно через змінні середовища, які зчитуються з файлу .env і не зберігаються безпосередньо в коді. Серверна частина валідує конфігурацію при старті за схемою: якщо обов'язкову змінну не задано або секретний ключ підпису токенів коротший за 32 символи, сервер не запускається й виводить зрозуміле повідомлення про помилку. Це переводить помилки конфігурації з категорії прихованих збоїв під час роботи в категорію явної відмови при запуску. Перелік параметрів конфігурації наведено в табл. 3.1.

Таблиця 3.1

Параметри конфігурації системи

Змінна оточення	Призначення
Database_Url	Рядок підключення до PostgreSQL
Redis_Url	Адреса сховища Redis
Port	Порт, на якому серверна частина очікує запити
Node_Env	Режим роботи: development або production
JWT_Secret	Секретний ключ підпису сесійних JWT-токенів (мінімум 32 символи)
Deadlock_API_Base_Url	Базова адреса зовнішнього ігрового API
Deadlock_API_Key	Ключ доступу до Deadlock API
Steam_API_Key	Ключ Steam Web API для отримання профілів
Steam_Realm	Домен, зареєстрований для авторизації через Steam OpenID
Steam_Return_Url	URL зворотного виклику після авторизації у Steam
Frontend_Url	Адреса клієнтської частини для налаштування політики CORS
Player_Cache_Ttl_Minutes	Час, протягом якого дані гравця вважаються свіжими
Redis_Cache_Ttl_Seconds	Час життя записів Кешу в Redis (у секундах)
Devs_Activity_Channel	Назва Telegram-каналу зі стрічкою активності розробників
Next_Public_API_Url	Адреса серверного API, на яку спрямовується клієнтська частина

За умови коректного виконання всіх кроків і задання параметрів платформа готова до роботи: серверна частина починає очікувати з'єднання на

визначеному порту, а клієнтська стає доступною у браузері за публічною адресою.

3.2. Структура інтерфейсу платформи для збору та аналізу ігрової статистики

Інтерфейс платформи побудовано як односторінковий застосунок (SPA) на Next.js із маршрутизацією за файловою структурою. Усі сторінки об'єднує спільний каркас: верхня навігаційна панель і адаптивна область контенту. Навігаційна панель містить логотип-посилання на головну сторінку, набір пунктів меню та блок авторизації (кнопку входу через Steam або аватар і ім'я авторизованого користувача). На вузьких екранах меню згортається у бургер-кнопку.

На рис. 3.2 зображено карту навігації платформи з усіма сторінками та зв'язками переходів між ними. Центральним вузлом виступає головна сторінка, з якої користувач потрапляє до профілю гравця, каталогів героїв і предметів, таблиці лідерів, розділу оновлень та стрічки активності розробників. Стрілки показують напрями переходів: із профілю можна відкрити сторінку матчу, з матчу — сторінки героїв, гравців і предметів, що формує наскрізну зв'язність між аналітичними екранами. Така структура дозволяє користувачеві рухатися від загального огляду до деталей будь-якого об'єкта без повернення на головну сторінку.

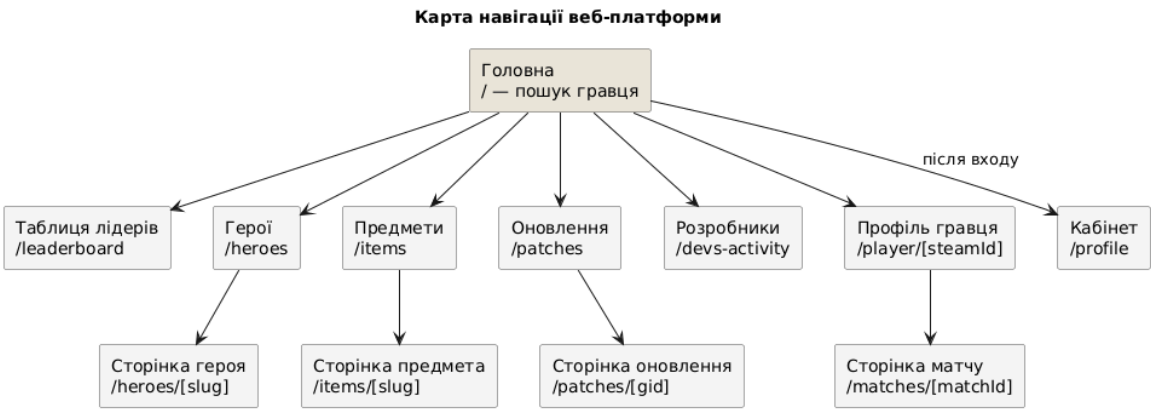


Рис. 3.2. Карта навігації вебплатформи

Початковою точкою взаємодії є головна сторінка, що відкривається одразу при заході на сайт (рис. 3.3). Її центральний елемент — пошуковий

рядок, у який користувач вводить ідентифікатор гравця у будь-якому з підтримуваних форматів: 17-значний SteamID64, короткий ідентифікатор облікового запису або повне посилання на профіль Steam. Нижче пошуку відображається віджет останніх знайдених гравців, що дозволяє швидко повернутися до раніше переглянутих профілів.

На рис. 3.3 зображено головну сторінку платформи у тому вигляді, в якому її бачить користувач одразу після заходу на сайт. Основну частину екрана займає пошуковий рядок із підказкою щодо підтримуваних форматів ідентифікатора, а під ним розташовано віджет нещодавно знайдених гравців з їхніми аватарами та іменами. Користувач вводить ідентифікатор і переходить до профілю, або одним натисканням відкриває один із недавніх профілів зі списку. Сторінка свідомо мінімалістична, щоб не відволікати від головного сценарію — пошуку гравця.

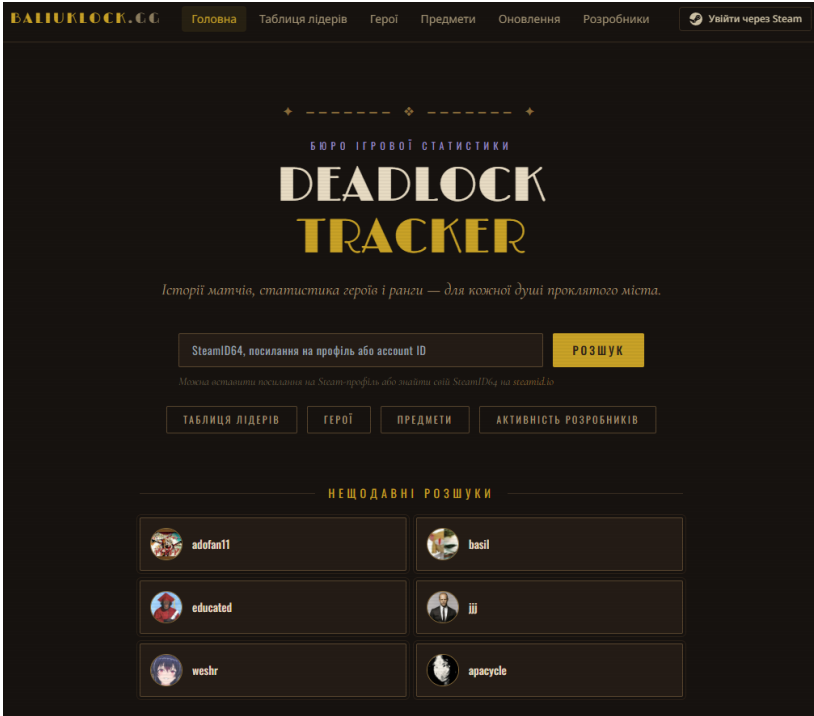


Рис. 3.3. Головна сторінка платформи

Сторінка профілю гравця є основним аналітичним екраном системи (рис. 3.4-7). Угорі розташовано зведений блок із аватаром, іменем, оцінкою поточного рангу та співвідношенням перемог і поразок. Нижче подано історію матчів із пагінацією, де кожен запис показує результат, обраного героя й

показники вбивств, смертей та асистів. Окрему секцію займає статистика за героями, відсортована за кількістю зіграних ігор, та персональні метрики ефективності з бейджами.

На рис. 3.4 зображено зведений блок профілю у верхній частині сторінки гравця. Тут показано аватар і нікнейм, узяті безпосередньо з облікового запису Steam, а поряд — узагальнену статистику: кількість перемог і поразок та кількість матчів, зіграних у режимі Street Brawl за останні 200 ігор. Цей блок дає користувачеві миттєве уявлення про загальну результативність гравця ще до перегляду детальної статистики. Він залишається у верхній частині екрана як орієнтир під час прокручування решти секцій профілю.

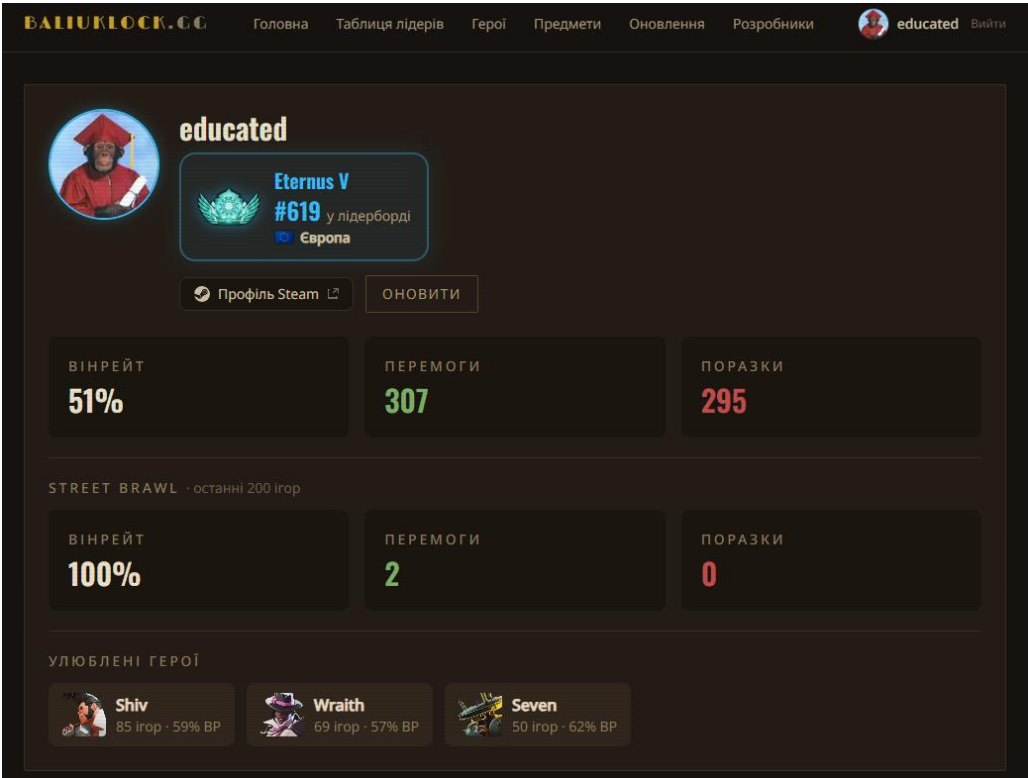


Рис. 3.4. Загальна інформація про гравця

На рис. 3.5 зображено секцію динаміки результатів гравця за останні 200 матчів. Календар активності відображає інтенсивність ігор за днями, а лінійний графік — прогрес або регрес у результатах із плином часу. Користувач може візуально оцінити періоди підйому чи спаду форми та регулярність ігрової активності. Така подача допомагає простежити загальну тенденцію, яку важко помітити за окремими матчами.



Рис. 3.5. Календар активності і графік історії ігор

На рис. 3.6 зображено секцію персональних метрик ефективності гравця. Зліва наведено усереднені показники за всю наявну історію, а праворуч — ті самі показники за останні 20 матчів, що дозволяє порівняти поточну форму із загальним рівнем. Метрики супроводжуються бейджами, які візуально виділяють сильні та слабкі сторони гри. Завдяки цьому користувач бачить не лише абсолютні значення, а й напрям зміни своєї результативності.



Рис. 3.6. Загальні показники гравця і за останні 20 матчів

На рис. 3.7 зображено історію матчів гравця з пагінацією. У кожному записі відображено режим гри, її тривалість і час завершення, а також ключову інформацію матчу: результат (перемога чи поразка), середній ранг лобі, обраного героя та коротку статистику гравця (вбивства, смерті, асисті). Кожен рядок є клікабельним і відкриває детальну сторінку відповідного матчу. Таким чином історія слугує точкою переходу від загального профілю до поглибленого аналізу конкретної гри.

BALIUKLOCK.CC		Головна	Таблиця лідерів	Герої	Предмети	Оновлення	Розробники	 educared	Вийти
Історія матчів 200		Статистика героїв							
L	V	Герой Abrams	2 / 12 / 11 1.08 KDA	Звичайний матч		24:08	1 дн тому		
W	V	Герой Bebop	3 / 10 / 20 2.30 KDA	Звичайний матч		39:15	1 дн тому		
L	V	Герой Bebop	1 / 12 / 9 0.83 KDA	Звичайний матч		29:49	1 дн тому		
L	V	Герой Vindicta	5 / 7 / 10 2.14 KDA	Звичайний матч		39:08	1 дн тому		
W	W	Герой Victor	9 / 4 / 9 4.50 KDA	Звичайний матч		24:55	2 дн тому		
W	W	Герой Venator	12 / 0 / 7 19.00 KDA	Звичайний матч		19:31	2 дн тому		
W	V	Герой Mirage	14 / 3 / 5 6.33 KDA	Звичайний матч		21:17	2 дн тому		
L	W	Герой Victor	15 / 9 / 5 2.22 KDA	Звичайний матч		29:50	2 дн тому		
L	IV	Герой Warden	0 / 1 / 0 0.00 KDA	Звичайний матч		9:58	10 дн тому		
W	IV	Герой Wraith	11 / 3 / 9 6.67 KDA	Звичайний матч		31:30	14 дн тому		
L	V	Герой Seven	2 / 4 / 7 2.25 KDA	Звичайний матч		32:35	14 дн тому		

Рис. 3.7. Історія матчів

Із профілю користувач переходить до детальної сторінки будь-якого матчу (рис. 3.8-11). На ній відображено склади обох команд із підсумковими показниками всіх дванадцяти учасників, а також інтерактивні графіки перебігу гри — динаміка зароблених коштів, вбивств і завданої шкоди в часі, — побудовані з метаданих повтору.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

На рис. 3.8 зображено верхню частину сторінки матчу із загальною інформацією та складами команд. Угорі вказано час, коли було зіграно матч, його тривалість і режим гри, а нижче — обидві команди з усіма учасниками, їхніми героями, зібраними предметами, короткою статистикою та рангами команд. Імена гравців, іконки героїв і предметів є клікабельними та ведуть на відповідні сторінки. Це робить картку матчу не лише підсумком гри, а й точкою входу до подальшого аналізу будь-якого її елемента.

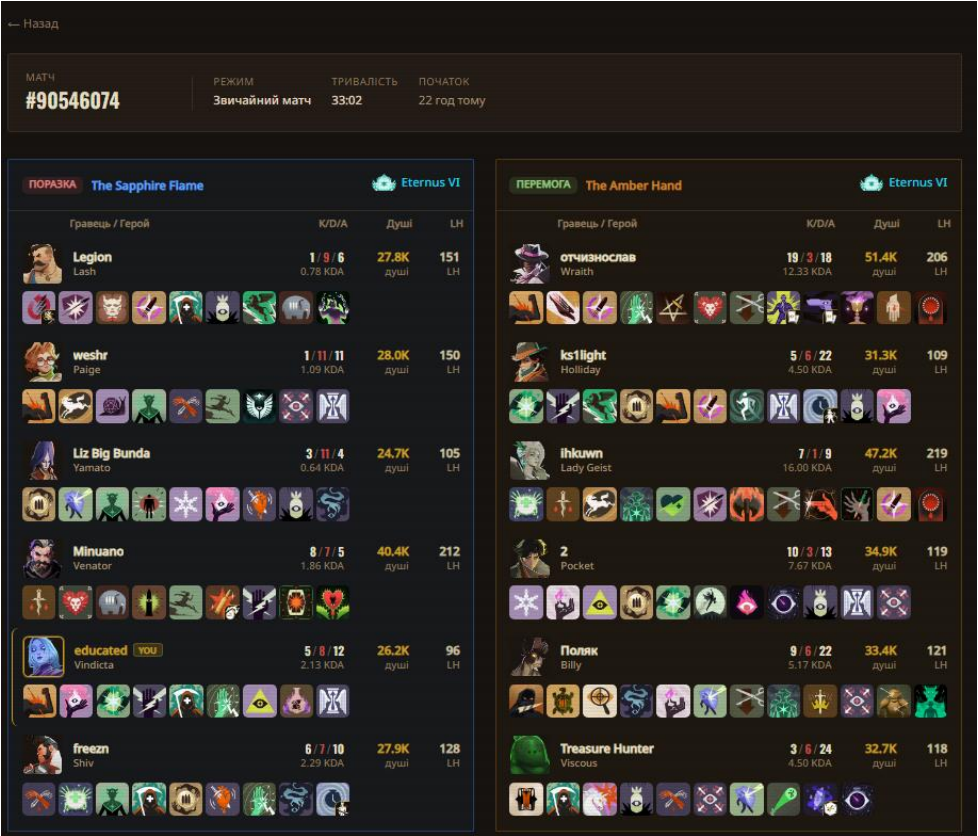


Рис. 3.8. Сторінка детального перегляду матчу

На рис. 3.9 зображено блок загальних графіків матчу. Графік нетворсу команд показує, у які проміжки домінувала кожна з команд, а також момент і команду, що знешкодила одного з ключових об'єктів — мідбоса. Блок аналітики відображає узагальнену статистику за обраними категоріями для конкретних ігрових проміжків, що допомагає визначити гравців із найбільшим впливом на гру в той чи інший період. Завдяки цьому користувач може зрозуміти не лише фінальний результат, а й динаміку переваги впродовж усього матчу.

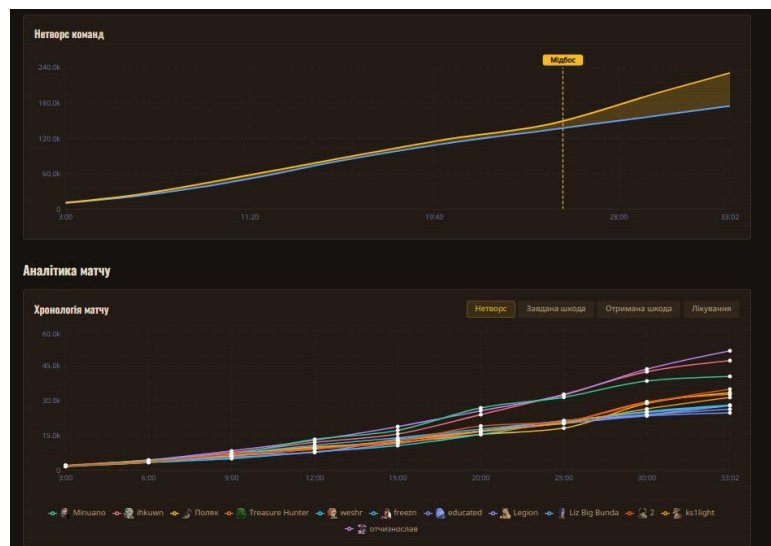


Рис. 3.9. Загальні графіки всіх гравців разом, та кожного окремо

На рис. 3.10 зображено основний аналітичний блок сторінки матчу, що відтворює прогрес обраного героя протягом гри. За допомогою повзунка користувач переміщується в часі, і разом із ним змінюються зібрані предмети гравця, рівні прокачаних здібностей та характеристики героя на відповідний момент. Поряд розташовано блок мапи, який детально показує переміщення героя по полю бою у вибраний проміжок. Цей інструмент дозволяє покроково простежити, як розвивався герой і де він перебував у кожен момент матчу.

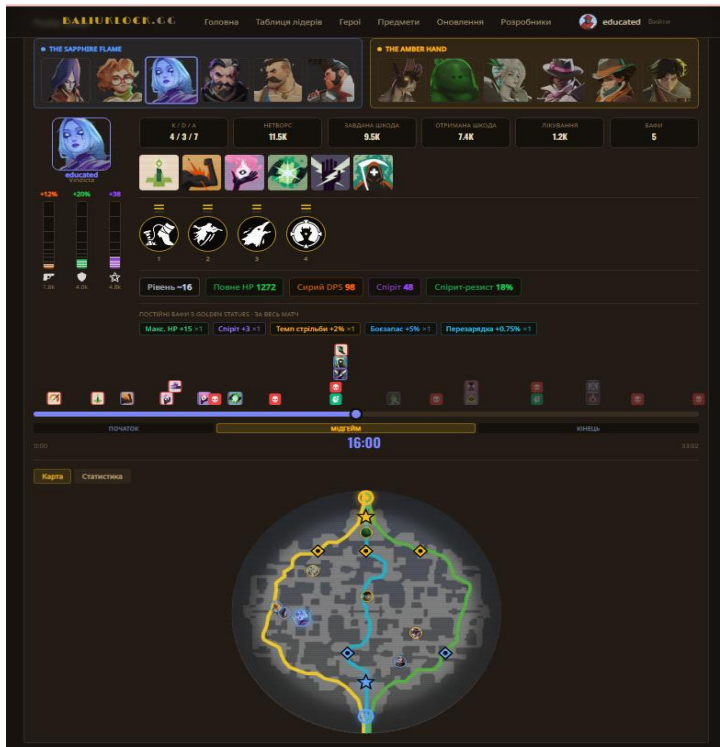


Рис. 3.10. Графік з хронологією предметів та рухом впродовж мапи

На рис. 3.11 зображено другий аналітичний блок сторінки матчу, який, на відміну від аналітичного блоку, який зображено на рис. 3.10, зосереджено на детальній статистиці окремого персонажа. Тут показано нанесену та отриману шкоду в розрізі кожного ворожого героя, а також основні джерела отримання душ (нетворсу). Дані так само синхронізовано з часовою шкалою матчу, тож їх можна переглядати в динаміці. Такий розріз допомагає зрозуміти, проти кого герой був ефективним і за рахунок чого набирав перевагу.



Рис. 3.11. Графік з хронологією предметів та статистики впродовж матчу

Каталог героїв подає сітку карток з усіма персонажами гри, кожна з яких містить іконку, ім'я та показники вінрейту й пік-рейту (рис. 3.12). Перехід до конкретного героя відкриває детальну сторінку (рис. 3.13-15), де відображено чотири сигнатурні здібності з рівнями прокачування та перекладеними

українською описами, панелі характеристик у стилі ігрового клієнта — зброя, живучість і дух — із позначенням статів, що масштабуються від сили духу, а також розподіл ефективності героя за ранговими групами.

На рис. 3.12 зображено сторінку з усіма персонажами гри, поданими у вигляді сітки карток. Кожна картка містить іконку та ім'я героя, а також коротку статистику: вінрейт, пік-рейт і загальну кількість матчів за участю персонажа. У верхній частині сторінки розташовано фільтр за ранговими групами та поле пошуку героя за іменем. Кожна картка є клікабельною й відкриває детальну сторінку відповідного героя.

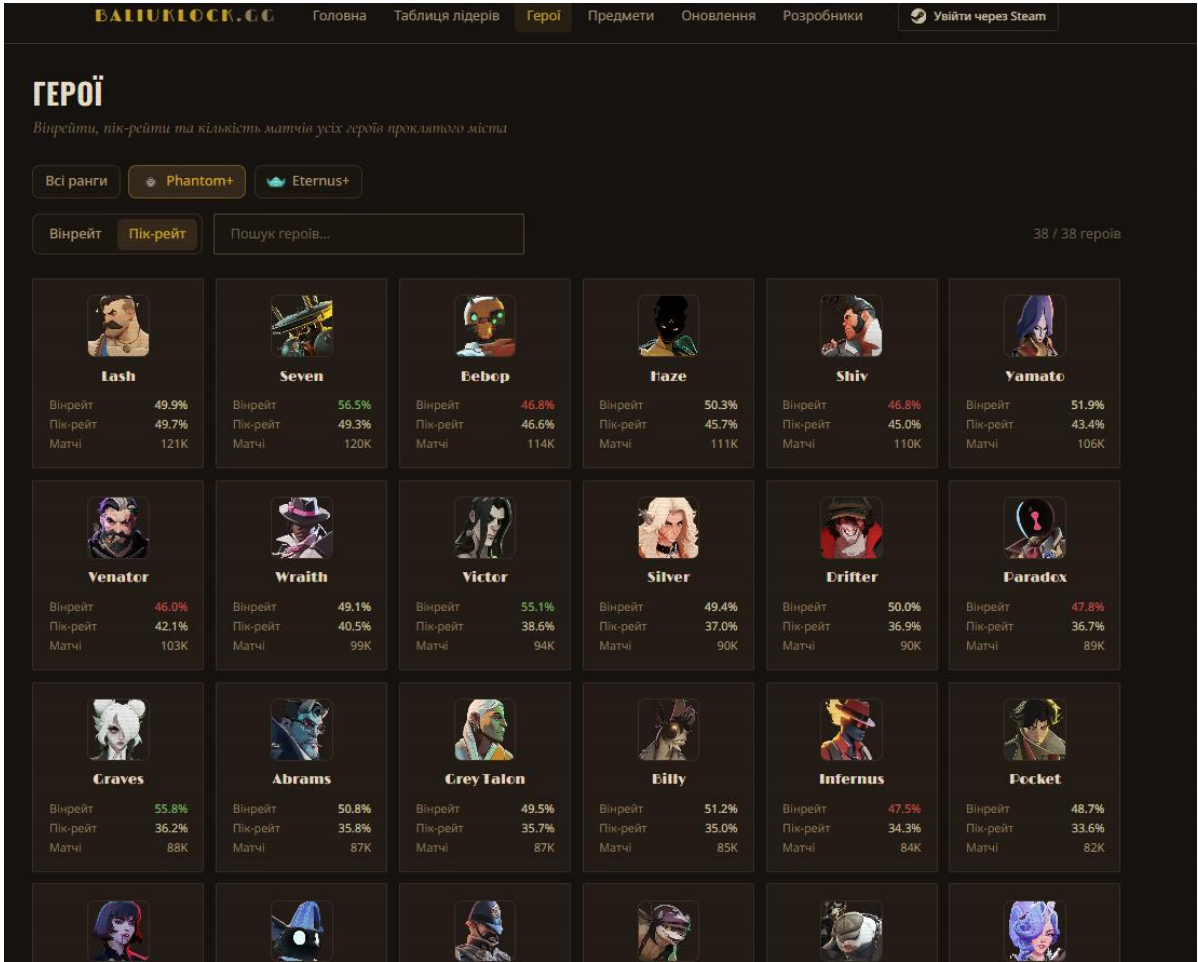


Рис. 3.12. Сторінка з всіма героями гри

На рис. 3.13 зображено детальну сторінку героя із загальною інформацією та панелями характеристик. Систему відображення статів перенесено безпосередньо з ігрового клієнта й розділено за категоріями — зброя, живучість і дух. Окремо позначено характеристики, що масштабуються

від сили духу, для яких реалізовано відповідну візуалізацію на платформі. Це дозволяє користувачеві оцінити параметри героя в тому ж вигляді, в якому він звик бачити їх у грі.

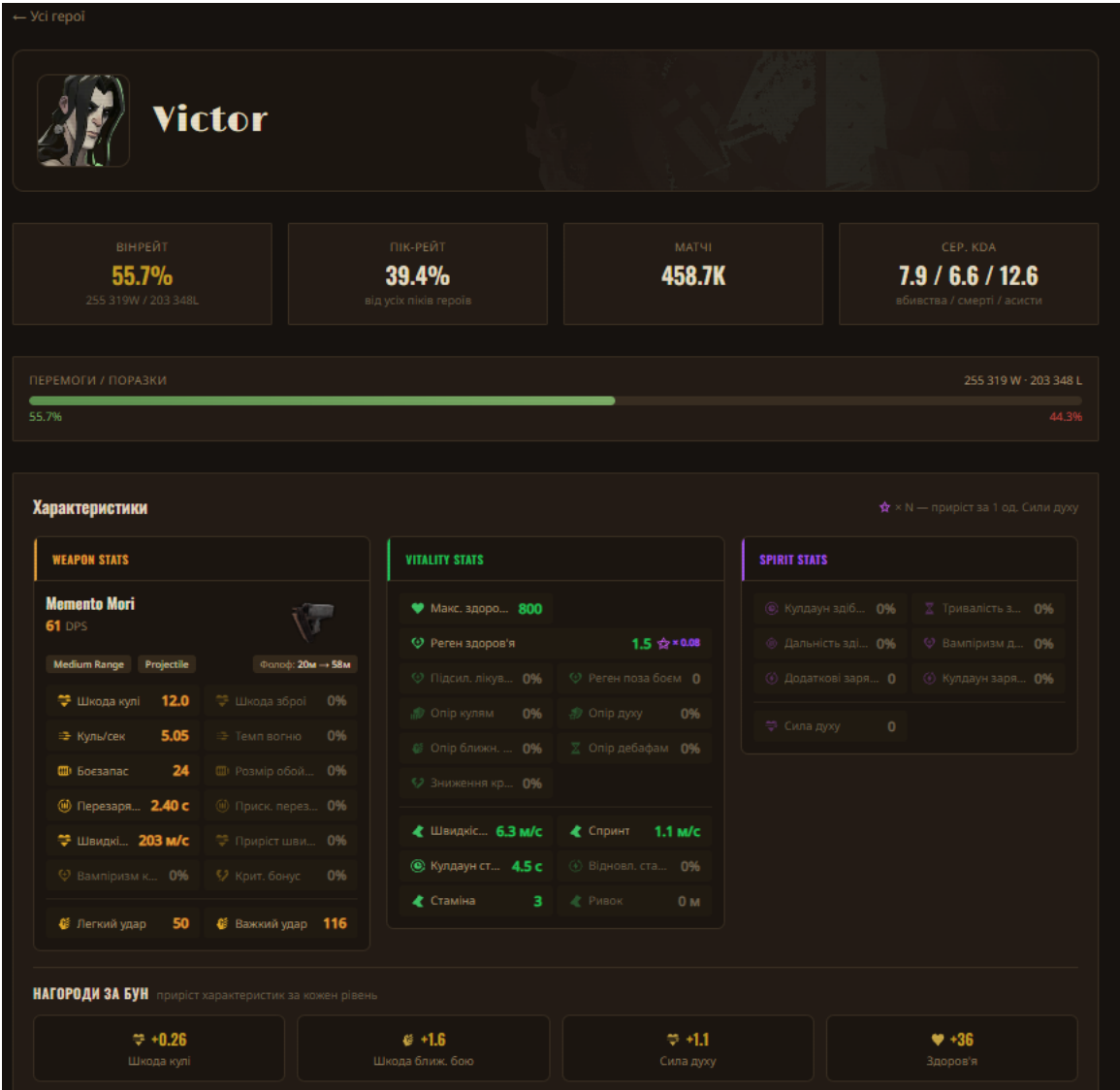


Рис. 3.13. Окрема сторінка героя з загальною інформацією та характеристиками

На рис. 3.14 зображено секцію з ігровою історією героя та його чотирма сигнатурними здібностями. Описи здібностей перекладено українською й супроводжено рівнями прокачування. Як історія, так і здібності беруться безпосередньо з даних гри, тому після оновлень і змін балансу інформація автоматично оновлюється й залишається актуальною. Завдяки цьому користувач завжди бачить чинні характеристики здібностей без потреби звертатися з ігровим клієнтом.

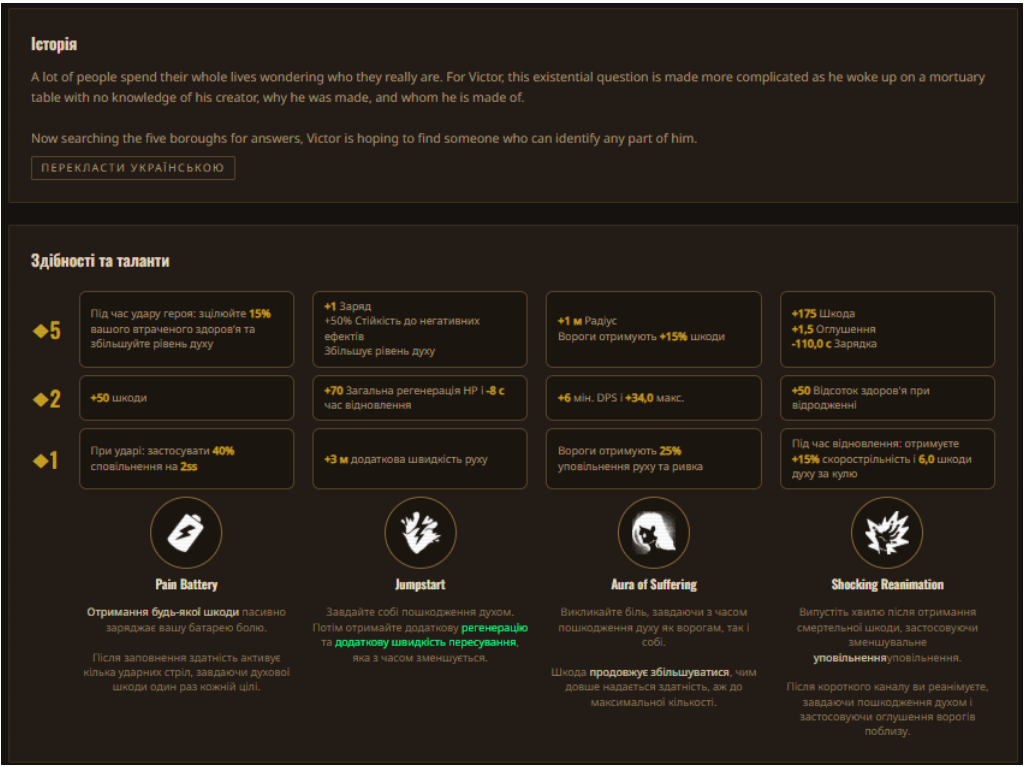


Рис. 3.14. Ігрова історія героя та його здібності

На рис. 3.15 зображено заключний блок сторінки героя — розподіл його ефективності за ранговими групами. Графік показує популярність і силу персонажа на різних рангах, від нижчих до найвищих. Це дає змогу побачити, на якому рівні гри герой є найсильнішим або найпопулярнішим. Така інформація корисна гравцям для вибору персонажів з огляду на власний ранг.

Результати за рангами			
Win rate across different skill tiers			
РАНГ	ВІНРЕЙТ	МАТЧІ	W / L
Initiate	56%	30 149	16 973 / 13 176
Seeker	57%	59 136	33 535 / 25 601
Alchemist	56%	38 720	21 782 / 16 938
Arcanist	56%	42 639	23 995 / 18 644
Ritualist	56%	48 073	26 979 / 21 094
Emissary	56%	59 152	32 909 / 26 243
Archon	56%	51 052	28 388 / 22 664
Oracle	55%	59 228	32 772 / 26 456
Phantom	55%	27 083	14 866 / 12 217
Ascendant	53%	32 251	17 239 / 15 012
Eternus	53%	11 184	5 881 / 5 303

Рис. 3.15. Статистика персонажа за рангами в грі

Каталог предметів організовано аналогічно й доповнено фільтрами за категорією та рівнем (рис. 3.16-18). Також є окрема категорія з пошуком

предметів за назвою (рис. 3.19). Детальна сторінка предмета містить його характеристики, статистику використання та дерево компонентів — з яких дешевших предметів він збирається (рис. 3.20).

На рис. 3.16 зображено каталог предметів категорії «Fairfax», що відповідає за підсилення зброї персонажа. Предмети подано сіткою карток із фільтрами за рівнем у верхній частині сторінки. Кожна картка є клікабельною й відкриває детальну сторінку відповідного предмета. Розподіл за категоріями допомагає користувачеві швидко знайти предмети потрібного напрямку збірки.



Рис. 3.16. Категорія Fairfax

На рис. 3.17 зображено каталог предметів категорії «MPS», яка відповідає за живучість персонажа. Структура сторінки повністю аналогічна попередній категорії: сітка карток із фільтрами за рівнем і клікабельними

переходами до деталей кожного предмета. Єдиний кольоровий акцент категорії відрізняє її від інших і відповідає колірному кодуванню самої гри. Це забезпечує послідовність інтерфейсу й полегшує орієнтування між категоріями.

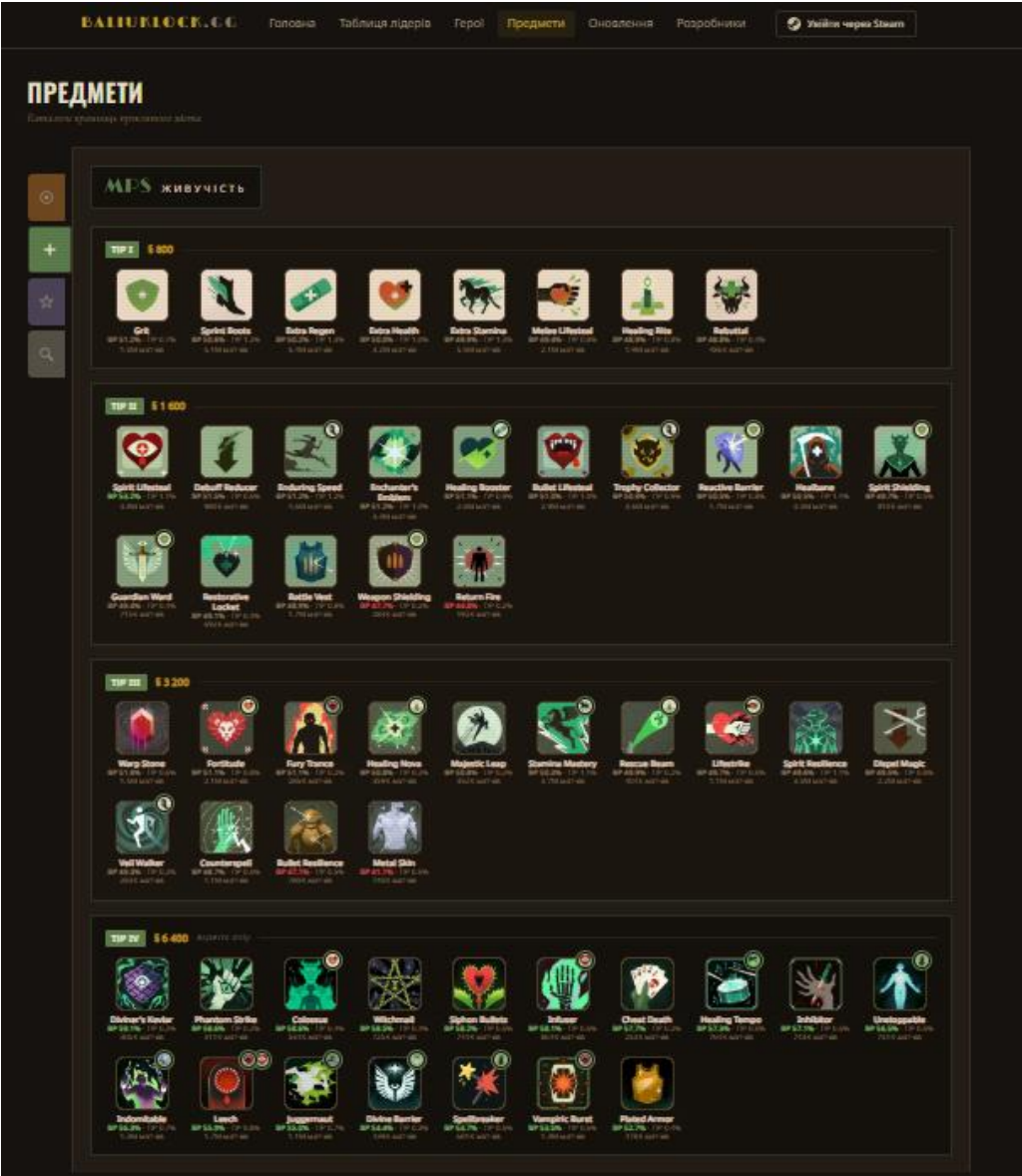


Рис. 3.17. Категорія MPS

На рис. 3.18 зображено каталог предметів категорії «Curiosity», що підсилює здібності героїв або надає активні предмети для протидії ворожим персонажам. Як і в інших категоріях, предмети подано сіткою клікабельних карток із фільтрами за рівнем. Перехід до картки відкриває детальний опис предмета з його ефектами. Завдяки однакової структурі всіх трьох категорій користувач взаємодіє з ними за єдиним звичним шаблоном.

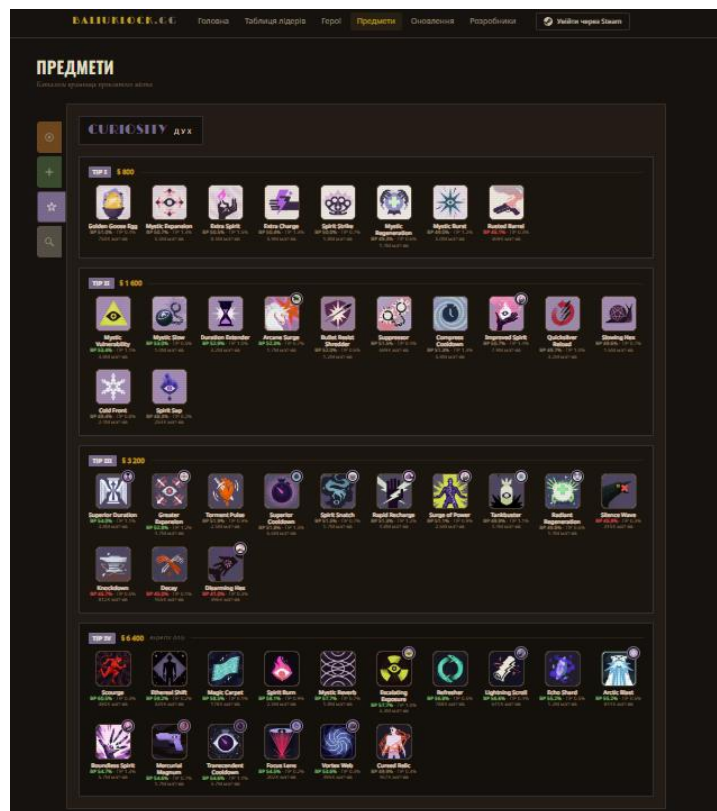


Рис. 3.18. Категорія Curiosity

На рис. 3.19 зображено режим пошуку предметів, який працює одразу за всіма категоріями. Користувач вводить назву в поле пошуку й отримує відповідні предмети у вигляді тих самих клікабельних карток. Цей режим зручний, коли точно відома назва предмета й немає потреби переглядати категорії вручну. Він доповнює перегляд категорій швидшим прямим доступом до конкретного предмета.

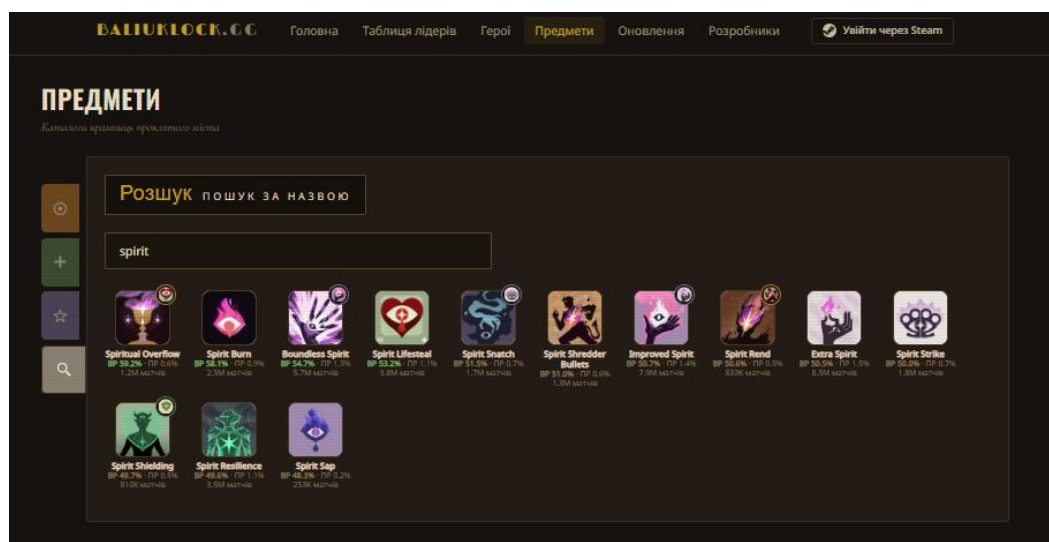


Рис. 3.19. Категорія пошуку предметів

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

На рис. 3.20 зображено детальну сторінку окремого предмета. Тут наведено його опис і характеристики, статистику використання, а також дерево компонентів — з яких дешевших предметів він збирається та у які дорожчі входить. Елементи дерева є клікабельними й дозволяють переходити між пов'язаними предметами. Це допомагає користувачеві зрозуміти повний шлях збірки предмета та його роль у загальній економіці гри.

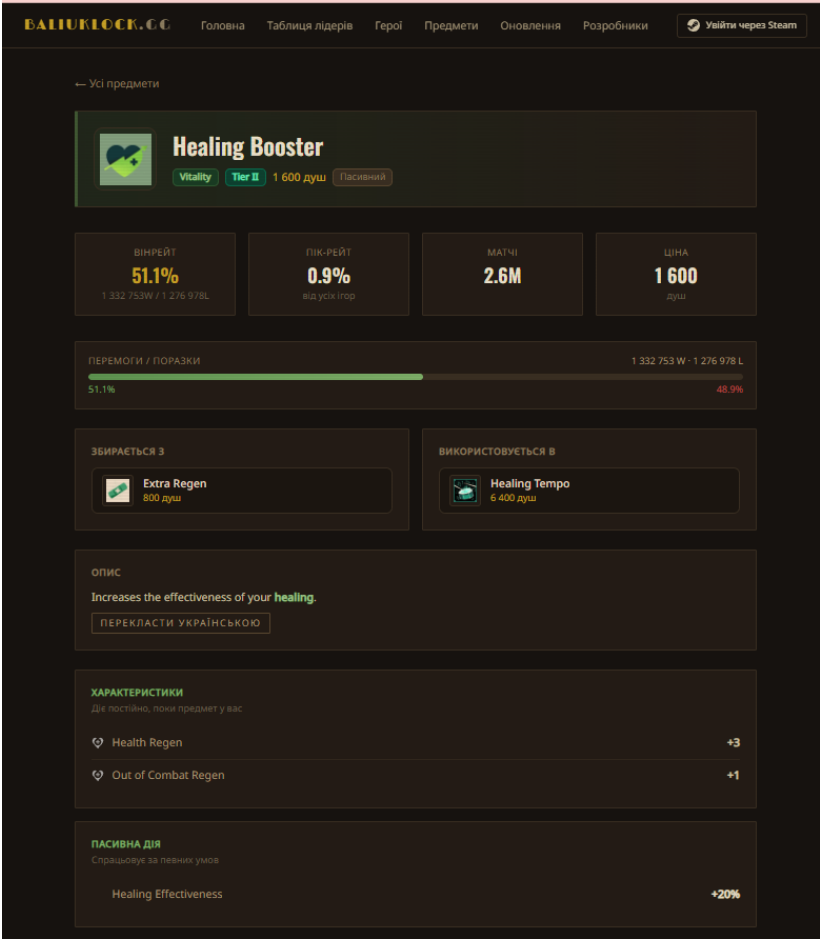


Рис. 3.20. Детальна сторінка предмета

Таблиця лідерів відображає рейтинги найкращих гравців із перемиканням між п'ятьма регіонами світу за допомогою кнопок угорі сторінки (рис. 3.21). Записи з однаковими позиціями коректно відображаються зі спільним номером, а імена впевнено розв'язаних гравців є клікабельними посиланнями на їхні профілі.

На рис. 3.21 зображено таблицю лідерів із рейтингом найкращих гравців. Угорі сторінки розташовано кнопки перемикання між п'ятьма регіонами світу, що оновлюють список відповідно до обраного регіону. Записи з однаковими

позиціями коректно відображаються зі спільним номером, а імена впевнено розв'язаних гравців є клікабельними посиланнями на їхні профілі. Таким чином таблиця слугує не лише рейтингом, а й точкою переходу до детального аналізу найсильніших гравців.

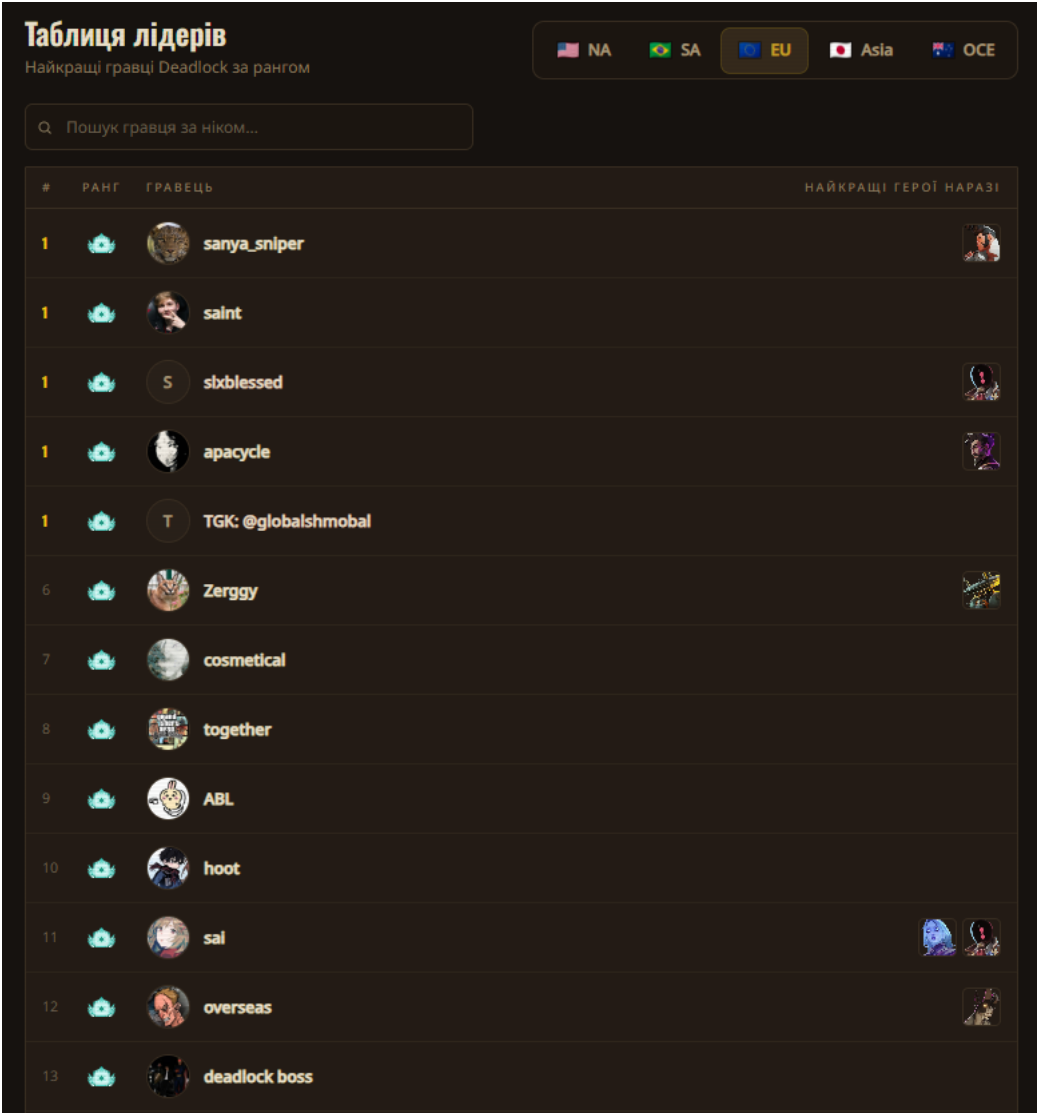


Рис. 3.21. Таблиця лідерів за регіонами

Розділ оновлень агрегує офіційні патч-ноути гри (рис. 3.22). Перелік відсортовано за датою, кожен запис позначено типом оновлення — дрібне, регулярне чи мажорне. Перехід до конкретного оновлення відкриває його повний текст, у якому зміни автоматично групуються у картки за героями та предметами: кожна така картка містить іконку, назву-посилання на відповідну сторінку та перелік змін, що значно полегшує читання довгих патч-ноутів (рис. 3.23).

На рис. 3.22 зображено перелік офіційних оновлень гри, відсортований за датою. Кожен запис позначено типом оновлення — дрібне, регулярне чи мажорне, — і ці типи візуально розділено кольором для кращого сприйняття. Користувач одразу бачить масштаб кожного патча й може обрати потрібний. Натискання на запис відкриває повний текст відповідного оновлення.

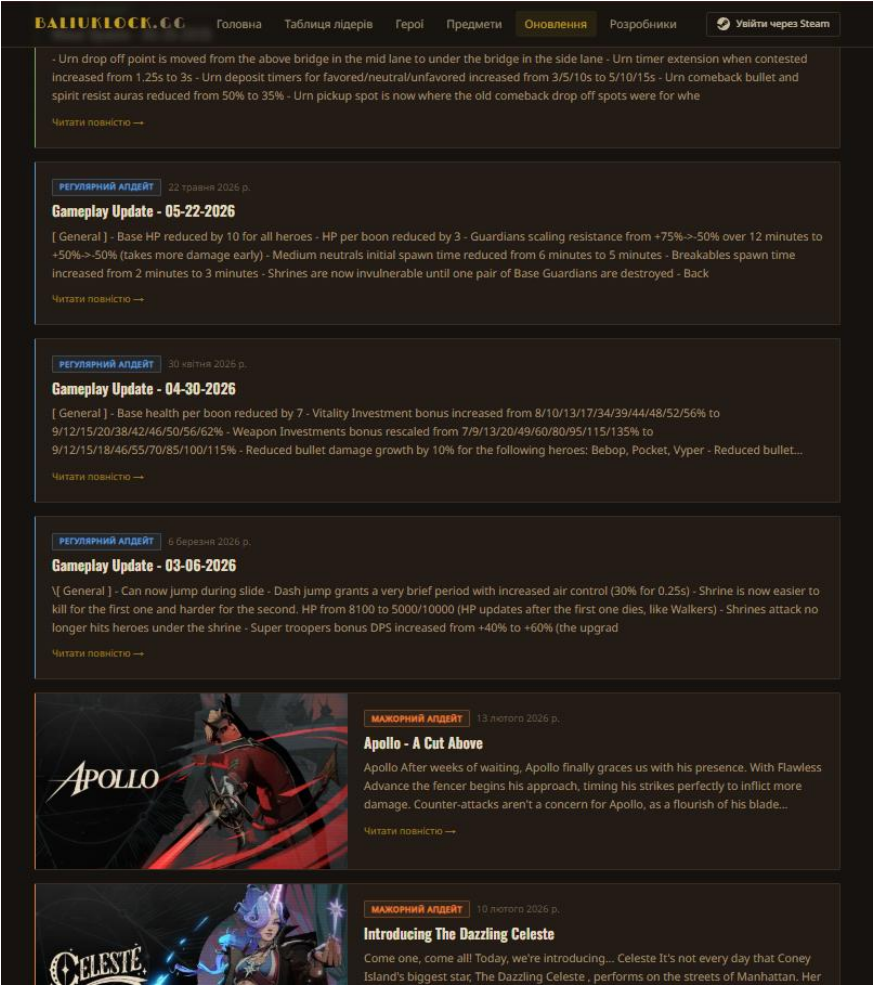


Рис. 3.22. Перелік оновлень гри

На рис. 3.23 зображено сторінку окремого оновлення з повним текстом патч-ноуту. Зміни автоматично групуються у картки за героями та предметами: кожна картка містить іконку, назву-посилання на відповідну сторінку та перелік змін, що стосуються цього об'єкта. Завдяки такому групуванню довгі патч-ноути читаються значно легше, ніж суцільним списком. Користувач може одразу перейти зі зміни на сторінку героя чи предмета, якого вона стосується.

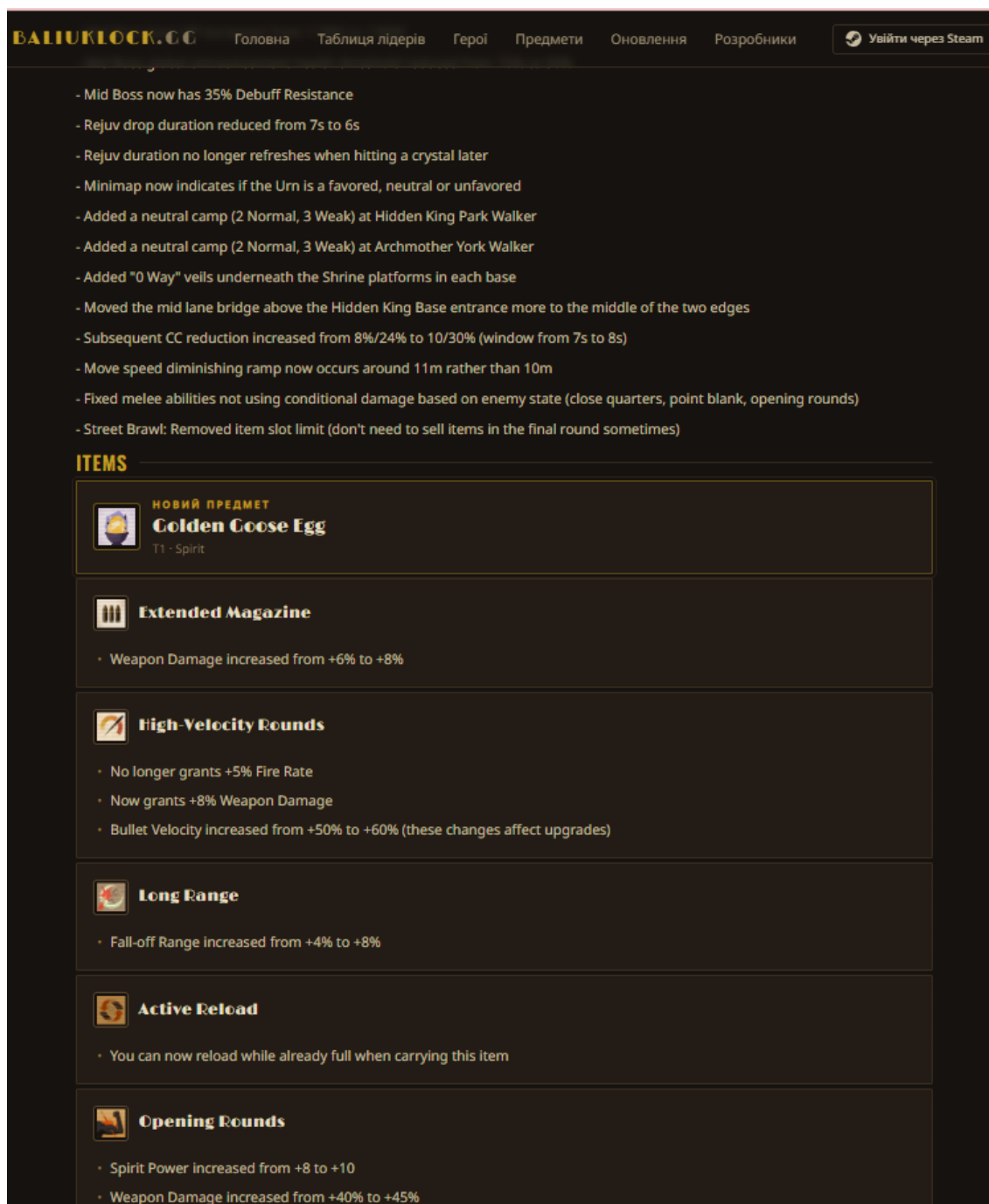


Рис. 3.23. Сторінка оновлення зі згрупованими змінами

На рис. 3.24 зображено стрічку активності розробників, що збирає їхні реальні дії на офіційному форумі у зворотному хронологічному порядку. Кожен запис показує ім'я розробника, його дію на форумі та час події у двох часових поясах. Дані надходять із Telegram-каналу спільноти, тож стрічка відображає актуальну активність майже в реальному часі. Це дозволяє користувачеві стежити за тим, над чим працюють розробники, не відвідуючи форум безпосередньо.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

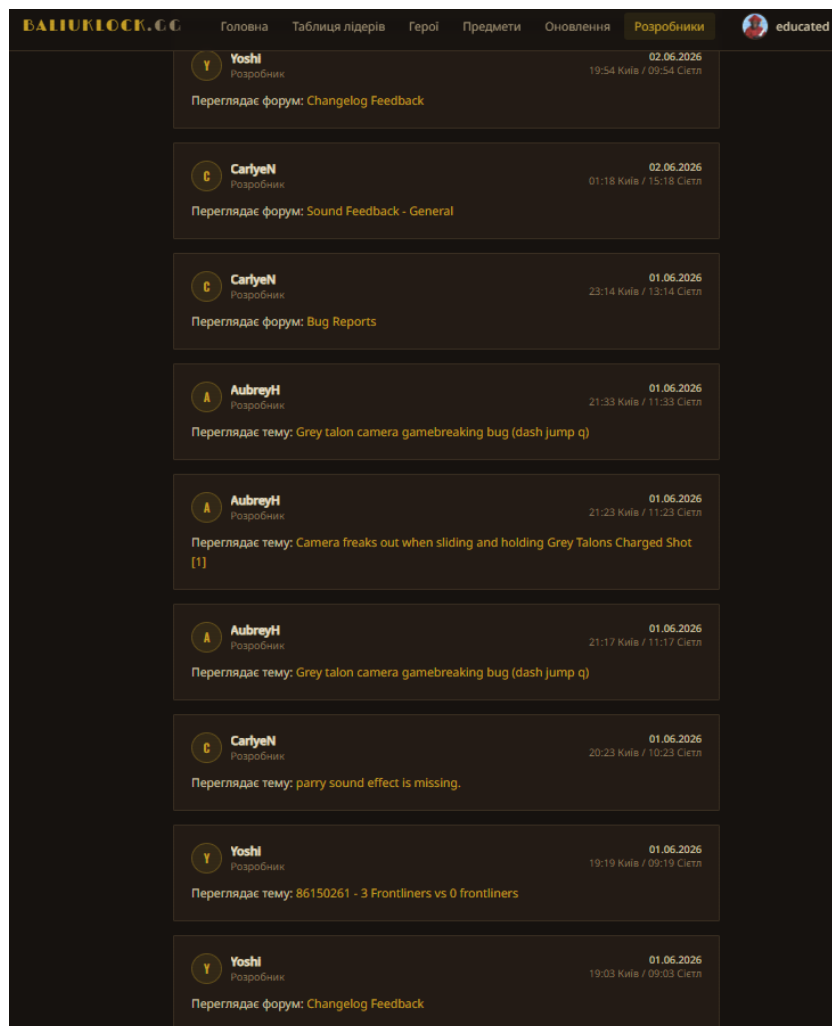


Рис. 3.24. Стрічка активності розробників

Авторизація виконується через обліковий запис Steam: натискання кнопки входу перенаправляє користувача на сторінку Steam, де він підтверджує особу, після чого повертається на платформу вже автентифікованим (рис. 3.25). Платформа не зберігає паролів — обліковий запис створюється автоматично при першому вході.

На рис. 3.25 зображено процес авторизації користувача через обліковий запис Steam. Після натискання кнопки входу користувача перенаправляє на офіційну сторінку Steam, де він підтверджує свою особу, а потім повертається на платформу вже автентифікованим. Платформа не отримує й не зберігає паролів — обліковий запис створюється автоматично при першому вході. Такий підхід забезпечує безпеку даних користувача й спрощує реєстрацію до одного натискання.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

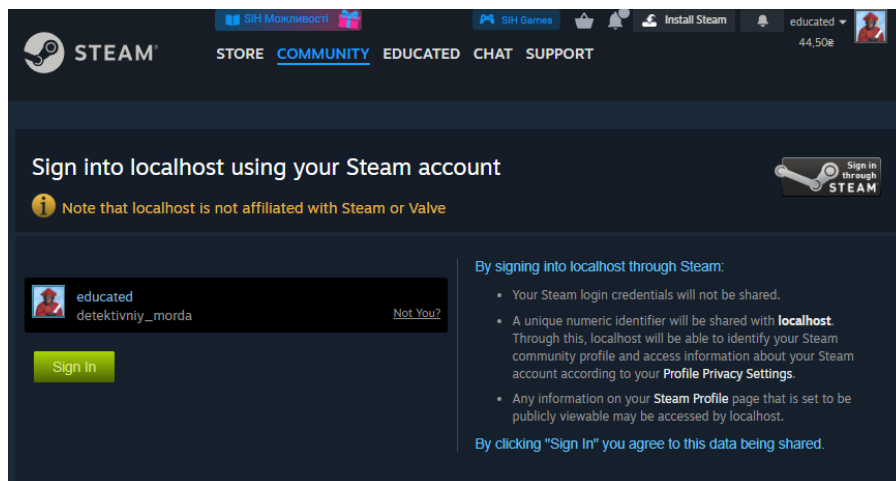


Рис. 3.25. Авторизація через Steam

Інтерфейс реалізовано в адаптивному дизайні за принципом mobile-first: сітки автоматично змінюють кількість колонок, другорядні колонки таблиць приховуються на вузьких екранах, а під час завантаження даних відображаються skeleton-заглушки, що повторюють форму майбутнього контенту [15]. Кольорову гаму виконано в темній ігровій стилістиці з акцентними кольорами для категорій характеристик, що відповідає колірному кодуванню самої гри. Описаний інтерфейс охоплює всі ключові сценарії використання платформи й дотримується принципів послідовності та мінімалізму: одні й ті самі візуальні шаблони повторюються на різних сторінках, що зменшує когнітивне навантаження і пришвидшує освоєння системи новим користувачем.

3.3. Тестування платформи

Тестування є невід'ємною складовою життєвого циклу розробки, оскільки саме воно дає змогу виявити дефекти та переконатися у відповідності продукту висунутим вимогам. Особливість тестування цієї платформи полягає в тому, що її коректність визначається не лише власною логікою, а й правильністю взаємодії з кількома зовнішніми сервісами. Тому свідомо обрано підхід інтеграційного та функціонального тестування: кожен сценарій проходить наскрізь — через клієнтську частину, серверну логіку, кешуюче сховище, базу даних і зовнішні API, — тож позитивний результат водночас підтверджує і коректність окремої логіки, і працездатність розгортання.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

Перед функціональною перевіркою проведено статичний аналіз типів. Оскільки клієнтська й серверна частини написані мовою TypeScript із наскрізним спільним пакетом типів, значний клас помилок виявляється ще на етапі компіляції: звернення до неіснуючого поля чи невідповідність типу між клієнтом і сервером спричиняють помилку компіляції, а не збій під час роботи. Складання всіх трьох пакетів виконується без помилок, що підтверджує типову коректність коду. Окремо виконано інтеграційну перевірку розгортання: піднято інфраструктуру в контейнерах, застосовано всі міграції бази даних і перевірено доступність серверного API та клієнтської частини.

Функціональне тестування проведено ручним методом за тест-кейсами — детально описаними наборами кроків з очікуваними результатами. Нижче наведено основні тест-кейси, що охоплюють ключовий функціонал платформи.

Перший тест-кейс перевіряє основний сценарій системи — пошук гравця за різними форматами ідентифікації та коректність відображення його статистики (табл. 3.2).

Таблиця 3.2

Тест-кейс 1: пошук і перегляд профілю гравця

№	Кроки	Очікуваний результат	Статус
1	Пошук за 17-значним SteamID64	Перехід на профіль з коректною статистикою	Виконано
2	Пошук за коротким ідентифікатором облікового запису	Нормалізація ідентифікатора та перехід на профіль	Виконано
3	Пошук за посиланням на Steam-профіль	Витягнення ідентифікатора з посилання та перехід	Виконано
4	Пошук за неіснуючим ідентифікатором	Повідомлення про відсутність гравця (HTTP 404)	Виконано
5	Перегляд історії матчів із перемиканням сторінок	Коректне завантаження сторінок пагінації	Виконано
6	Повторне відкриття профілю	Миттєва відповідь із кешу без звернення до API	Виконано

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

Другий тест-кейс охоплює перегляд каталогів героїв і предметів та деталізацію (табл. 3.3).

Таблиця 3.3

Тест-кейс 2: каталоги та деталізовані сторінки

№	Кроки	Очікуваний результат	Статус
1	Відкриття каталогу героїв	Сітка карток зі статистикою вінрейту й пікрейту	Виконано
2	Перехід на сторінку героя	Здібності, панелі характеристик, розподіл за рангами	Виконано
3	Перевірка перекладу описів здібностей	Описи відображаються українською мовою	Виконано
4	Відкриття каталогу предметів і застосування фільтрів	Список звужується за категорією та рівнем	Виконано
5	Перехід на сторінку предмета	Характеристики та дерево компонентів	Виконано

Третій тест-кейс присвячено таблиці лідерів і перегляду матчу (табл. 3.4).

Таблиця 3.4

Тест-кейс 3: таблиця лідерів і детальний матч

№	Кроки	Очікуваний результат	Статус
1	Перемикання регіонів у таблиці лідерів	Завантаження рейтингу обраного регіону	Виконано
2	Перевірка гравців з однаковими позиціями	Спільний номер позиції відображається коректно	Виконано
3	Перехід на профіль гравця з лідерборду	Клік по впевнено розв'язаному гравцю відкриває профіль	Виконано
4	Відкриття детальної сторінки матчу	Склади команд і показники дванадцяти гравців	Виконано
5	Перегляд графіків перебігу гри	Графіки нетворсу, вбивств і шкоди побудовано з повтору	Виконано

Четвертий тест-кейс перевіряє авторизацію через Steam та керування сесією (табл. 3.5).

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

Таблиця 3.5

Тест-кейс 4: авторизація та сесія

№	Кроки	Очікуваний результат	Статус
1	Натискання кнопки входу через Steam	Перенаправлення на сторінку Steam OpenID	Виконано
2	Підтвердження особи у Steam	Повернення на платформу, видача сесії, поява профілю в меню	Виконано
3	Запит без сесії до захищеного ендпоінта	Відповідь HTTP 401	Виконано
4	Вихід із системи	Очищення сесійного cookie, повернення у стан гостя	Виконано

П'ятий тест-кейс перевіряє нефункціональні властивості платформи — її стійкість до збоїв зовнішніх залежностей та поведінку під підвищеним навантаженням (табл. 3.6). На відміну від попередніх сценаріїв, що підтверджували коректність логіки платформи, цей кейс зосереджено на здатності системи зберігати працездатність у несприятливих умовах.

Таблиця 3.6

Тест-кейс 5: стійкість системи до збоїв та обмеження навантаження

№	Кроки	Очікуваний результат	Статус
1	Перевищення 100 запитів за хвилину з однієї IP	Відповідь HTTP 429 (обмеження частоти)	Виконано
2	Зупинка контейнера Redis	Платформа залишається працездатною (деградація лише у швидкості)	Виконано
3	Тимчасова недоступність зовнішнього API (429/503)	Автоматичні повторні спроби, відсутність відмови користувачеві	Виконано
4	Одночасні запити до того самого профілю	Перебудову виконує один процес, решта отримує наявний знімок	Виконано

Шостий тест-кейс перевіряє адаптивність інтерфейсу та коректність обробки контенту (табл. 3.7).

Тест-кейс 6: адаптивність інтерфейсу та коректність обробки контенту

№	Кроки	Очікуваний результат	Статус
1	Відкриття платформи на мобільному екрані	Згортання меню в бургер, перебудова сіток в одну колонку	Виконано
2	Перегляд таблиці на вузькому екрані	Приховування другорядних колонок	Виконано
3	Перегляд оновлення гри	Зміни героїв і предметів згруповано в окремі картки з іконками	Виконано
4	Перевірка стрічки розробників	Відображення лише офіційних повідомлень із префіксом «Developer:»	Виконано

Отже, у ході тестування перевірено всі основні сценарії роботи платформи, у яких могли б виникнути непередбачувані ситуації. За результатами перевірки встановлено, що система та її окремі компоненти працюють коректно, стійкі до типових збоїв зовнішніх сервісів і відповідають висунутим вимогам. Виявлені у процесі розробки дефекти — зокрема некоректне відображення статус-іконок у описах здібностей, помилки конвертації одиниць вимірювання та відсутність групування предметів у патч-нотах поза секцією предметів — було усунено.

Висновки до третього розділу

У третьому розділі розглянуто інтерфейс вебплатформи та порядок роботи з нею.

У першому підрозділі описано порядок встановлення й налаштування параметрів системи, а за допомогою діаграми розгортання детально розкрито архітектуру платформи: розподіл служб на контейнери Docker, взаємодію клієнтського браузеру, реверс-проксі, серверної частини, бази даних, кешу та зовнішніх сервісів. Показано два режими розгортання — для розробки й для продакшну — та описано керування системою виключно через змінні середовища з валідацією конфігурації при старті.

У другому підрозділі розглянуто структуру інтерфейсу платформи. Інтерфейс розроблено з урахуванням принципів зручності та інтуїтивності, що дозволяє користувачам легко орієнтуватися в системі без спеціальної підготовки. Описано всі ключові екрани — головну сторінку з пошуком, профіль гравця, детальний матч, каталоги героїв і предметів, таблицю лідерів, розділ оновлень та стрічку розробників, — а також адаптивну поведінку інтерфейсу на пристроях різного розміру.

У третьому підрозділі проведено тестування роботи програмного комплексу. Перевірку виконано трьома взаємодоповнювальними методами — статичним аналізом типів, інтеграційною перевіркою розгортання та функціональним ручним тестуванням за тест-кейсами, що охоплюють пошук і перегляд профілів, каталоги, таблицю лідерів, перегляд матчів, авторизацію, стійкість до збоїв та адаптивність. За результатами тестування підтверджено коректну роботу системи, її стійкість до типових збоїв зовнішніх сервісів і готовність до використання.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досягнуто її основної мети — розроблено та реалізовано веборієнтовану платформу для збору й аналізу ігрової статистики гри Deadlock, що агрегує дані з відкритих джерел і подає їх у зручному для аналізу вигляді. Робота складалася з кількох послідовних етапів, які охопили аналіз предметної області, проєктування, програмну реалізацію й тестування системи.

У першому розділі проаналізовано напрями використання інформаційних технологій для побудови платформ ігрової аналітики та визначено призначення майбутньої системи. Розглянуто три наявні аналоги — сервіси Tracklock.gg, Lockblaze.com і Statlocker.gg, — що дозволило виявити їхні сильні та слабкі сторони. Установлено, що жоден з аналогів не поєднує повний безоплатний функціонал, якісну адаптацію під мобільні пристрої та агрегацію всієї інформації про гру в одному місці, а спільним обмеженням усіх розглянутих рішень є відсутність україномовної локалізації. Це обґрунтувало доцільність розробки власної платформи. За результатами порівняння архітектурних підходів обрано клієнт-серверну модель з організацією кодової бази у вигляді монорепозиторію зі спільним пакетом типів та шаровим поділом серверної частини, а як технологічний стек — наскрізну мову TypeScript, фреймворк Next.js (React) для клієнта, середовище Node.js із фреймворком Express для сервера, реляційну СУБД PostgreSQL з ORM Prisma та сховище Redis для кешування.

У другому розділі виконано детальне проєктування та програмну реалізацію системи. На основі аналізу вимог визначено модель доступу з двох станів — гостя й авторизованого через Steam користувача — та побудовано діаграму варіантів використання, що розмежовує доступний функціонал. Розроблено структуру бази даних із семи таблиць, подану у вигляді ER-діаграми та описану на рівні окремих таблиць із зазначенням полів, типів і ключів; схему свідомо побудовано компактною за принципом мінімізації даних, тоді як великі довідники та мінливу статистику винесено у кешуюче

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

сховище. Спроектовано й описано ключові алгоритми роботи системи, серед яких конвертація ідентифікаторів Steam, побудова знімків профілів із розподіленими блокуваннями, двофазне розв'язання неоднозначних записів таблиці лідерів зі скоринговим зіставленням, розбір стиснутих файлів повторів матчів та кешований переклад ігрових текстів. Реалізацію серверної частини виконано засобами Node.js та Express із автентифікацією через протокол Steam OpenID 2.0 і веденням сесії на основі stateless JWT-токена в захищеному cookie, а клієнтську частину — засобами Next.js із керуванням серверним станом через бібліотеку TanStack Query.

У третьому розділі розглянуто інтерфейс системи та порядок роботи з нею. За допомогою діаграми розгортання описано фізичну архітектуру платформи — розподіл служб на контейнери Docker, об'єднані реверс-проксі nginx, — та порядок її встановлення й налаштування через змінні середовища з валідацією конфігурації при старті. Докладно проаналізовано структуру інтерфейсу, який побудовано з урахуванням принципів зручності та інтуїтивності й дозволяє користувачам легко орієнтуватися в системі без спеціальної підготовки. Проведено тестування програмного комплексу трьома взаємодоповнювальними методами — статичним аналізом типів, інтеграційною перевіркою розгортання та функціональним ручним тестуванням за тест-кейсами, — у ході якого перевірено всі ключові процеси системи, зокрема пошук і перегляд профілів гравців, перегляд матчів із графіками, каталоги героїв і предметів, таблицю лідерів, авторизацію, стійкість до збоїв зовнішніх сервісів та адаптивність інтерфейсу. Результати тестування підтвердили коректну роботу основного функціоналу та надійність розробленої системи.

Окремою інженерною особливістю платформи є побудова архітектури навколо головного обмеження предметної області — лімітів зовнішніх API на частоту запитів. Усю взаємодію із зовнішніми сервісами зосереджено на сервері, який агрегує «сирі» дані й обслуговує багатьох користувачів з одного закешованого результату, а багаторівнева система кешування з

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		79

диференційованими термінами життя та розподіленими блокуваннями забезпечує швидкий відгук і стійкість до тимчасової недоступності джерел. Платформа має адаптивний інтерфейс у темній ігровій стилістиці з повною україномовною локалізацією, зокрема автоматичним перекладом описів здібностей героїв, що робить її доступною для україномовної аудиторії гравців.

Таким чином, у роботі реалізовано всі необхідні етапи створення працездатної веборієнтованої платформи для збору й аналізу ігрової статистики, що відповідає поставленим вимогам. Отримані результати мають практичну цінність і можуть бути використані гравцями спільноти для відстеження власного прогресу та обґрунтованого вибору героїв і предметів, а також становлять основу для подальшого вдосконалення системи й розширення її функціональних можливостей — зокрема додавання синергій персонажів, інтерактивної мапи та поглиблення персональної аналітики.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vanderkam D. Effective TypeScript: 83 Specific Ways to Improve Your TypeScript. 2nd ed. Sebastopol : O'Reilly Media, 2024. 322 p.
2. Casciaro M., Mammino L. Node.js Design Patterns. 3rd ed. Birmingham : Packt Publishing, 2020. 660 p.
3. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 310 p.
4. Obe R. O., Hsu L. S. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 4th ed. Sebastopol : O'Reilly Media, 2025. 314 p.
5. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
6. TypeScript Documentation. Microsoft. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 10.06.2026).
7. Node.js Documentation. OpenJS Foundation. URL: <https://nodejs.org/docs/latest-v20.x/api/> (дата звернення: 10.06.2026).
8. Express — Fast, unopinionated, minimalist web framework for Node.js. URL: <https://expressjs.com/> (дата звернення: 10.06.2026).
9. Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 10.06.2026).
10. React Documentation. Meta. URL: <https://react.dev/> (дата звернення: 10.06.2026).
11. Prisma ORM Documentation. URL: <https://www.prisma.io/docs> (дата звернення: 10.06.2026).
12. PostgreSQL 16 Documentation. PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/16/index.html> (дата звернення: 10.06.2026).
13. Redis Documentation. Redis Ltd. URL: <https://redis.io/docs/latest/> (дата звернення: 10.06.2026).
14. TanStack Query Documentation. URL: <https://tanstack.com/query/latest> (дата звернення: 10.06.2026).

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

15. Tailwind CSS Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 10.06.2026).
16. Recharts — A composable charting library built on React components. URL: <https://recharts.org/> (дата звернення: 10.06.2026).
17. Docker Documentation. Docker Inc. URL: <https://docs.docker.com/> (дата звернення: 10.06.2026).
18. nginx Documentation. URL: <https://nginx.org/en/docs/> (дата звернення: 10.06.2026).
19. Steamworks Web API Reference. Valve Corporation. URL: <https://partner.steamgames.com/doc/webapi> (дата звернення: 10.06.2026).
20. OpenID Authentication 2.0 — Final. OpenID Foundation, 2007. URL: https://openid.net/specs/openid-authentication-2_0.html (дата звернення: 10.06.2026).
21. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. Internet Engineering Task Force, 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 10.06.2026).
22. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: doctoral dissertation. University of California, Irvine, 2000. 162 p. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (дата звернення: 10.06.2026).
23. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format: RFC 8259. Internet Engineering Task Force, 2017. URL: <https://datatracker.ietf.org/doc/html/rfc8259> (дата звернення: 10.06.2026).
24. Protocol Buffers Documentation. Google. URL: <https://protobuf.dev/> (дата звернення: 10.06.2026).
25. Zod — TypeScript-first schema validation. URL: <https://zod.dev/> (дата звернення: 10.06.2026).
26. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, 2017. 616 p.

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

27. Carlson J. L. Redis in Action. Shelter Island : Manning Publications, 2013. 320 p.
28. Poulton N. Docker Deep Dive. 2nd ed. Birmingham : Packt Publishing, 2020. 350 p.
29. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 533 p.
30. Barth A. HTTP State Management Mechanism: RFC 6265. Internet Engineering Task Force, 2011. URL: <https://datatracker.ietf.org/doc/html/rfc6265> (дата звернення: 10.06.2026).
31. Deadlock API — Community-driven API for Deadlock game data. URL: <https://deadlock-api.com/> (дата звернення: 10.06.2026).

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

ДОДАТКИ

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		84



Рис. А.1 — Загальний алгоритм роботи з платформою

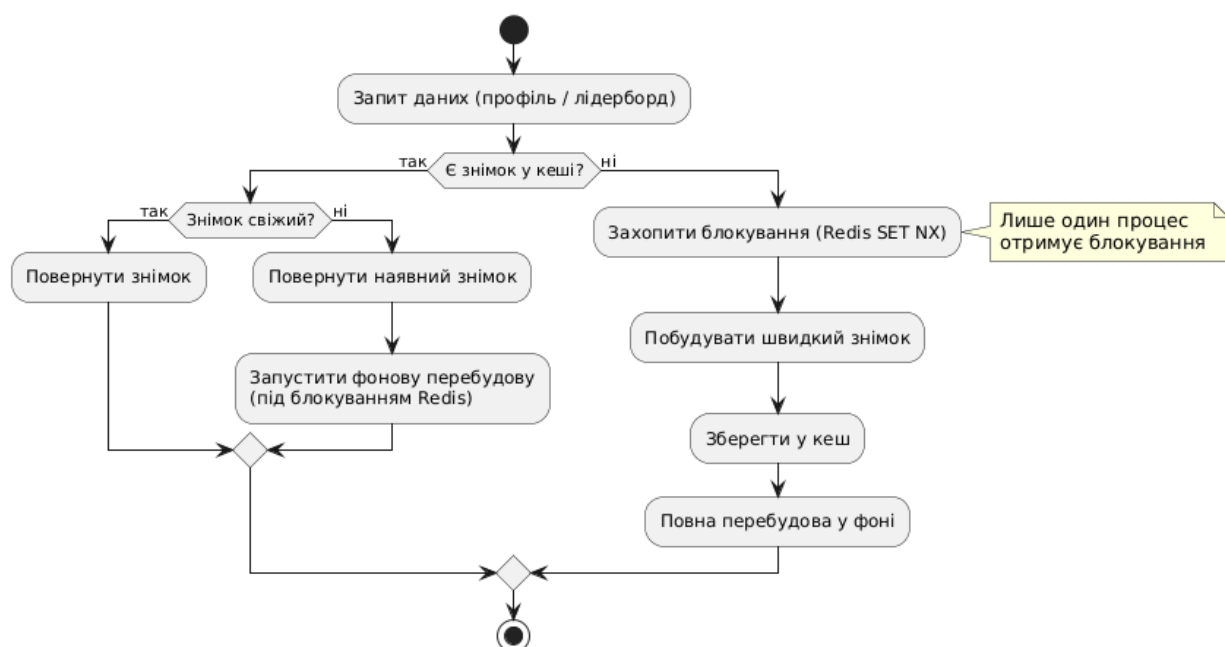


Рис. А.2 — Механізм зніmkів і багаторівневого кешування

Додаток Б

index.ts

```

import { z } from "zod";
const envSchema = z.object({
  NODE_ENV: z.enum(["development", "production", "test"]).default("development"),
  PORT: z.string().default("3001"),
  DATABASE_URL: z.string(),
  REDIS_URL: z.string().default("redis://localhost:6379"),
  JWT_SECRET: z.string().min(32),
  DEADLOCK_API_BASE_URL: z.string().default("https://api.deadlock-api.com"),
  DEADLOCK_API_KEY: z.string().optional(),
  STEAM_API_KEY: z.string().optional(),
  STEAM_REALM: z.string().default("http://localhost:3001"),
  STEAM_RETURN_URL: z.string().default("http://localhost:3001/api/auth/steam/callback"),
  FRONTEND_URL: z.string().default("http://localhost:3000"),
  PLAYER_CACHE_TTL_MINUTES: z.string().default("10"),
  REDIS_CACHE_TTL_SECONDS: z.string().default("300"),
});
const parsed = envSchema.safeParse(process.env);
if (!parsed.success) {

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

console.error("❌ Invalid environment variables:");
console.error(parsed.error.flatten().fieldErrors);
process.exit(1);
}
export const config = {
...parsed.data,
port: parseInt(parsed.data.PORT, 10),
playerCacheTtlMs: parseInt(parsed.data.PLAYER_CACHE_TTL_MINUTES, 10) * 60 * 1000,
redisCacheTtl: parseInt(parsed.data.REDIS_CACHE_TTL_SECONDS, 10),
};

```

streamId.ts

```

const STEAM_BASE = 76561197960265728n;
export function isSteamId64(id: string): boolean {
try {
return BigInt(id) > STEAM_BASE;
} catch {
return false;
}
}
export function steamId64ToAccountId(steamId64: string | bigint): number {
return Number(BigInt(steamId64) - STEAM_BASE);
}
export function accountIdToSteamId64(accountId: number): string {
return String(STEAM_BASE + BigInt(accountId));
}
export function normalizeToAccountId(id: string | number): number {
const str = String(id);
if (isSteamId64(str)) return steamId64ToAccountId(str);
return parseInt(str, 10);
}
export function canonicalSteamKey(id: string | number): string {
return String(normalizeToAccountId(id));
}

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

redis.ts

```
import { createClient } from "redis";
import { config } from "../index";
import logger from "../lib/logger";
const PREFIX = "dltracker:";

let _gaveUp = false;
const redis = createClient({
  url: config.REDIS_URL,
  socket: {
    reconnectStrategy: (retries) => {
      if (_gaveUp) return false;
      return Math.min(retries * 50, 500);
    },
  },
});

export async function connectRedis(): Promise<void> {
  try {
    await Promise.race([
      redis.connect(),
      new Promise<never>((_, reject) =>
        setTimeout(() => reject(new Error("Redis connect timeout")), 3_000),
      ),
    ]);
    logger.info("Redis connected");
  } catch (err) {
    _gaveUp = true;
    try { redis.disconnect(); } catch { }
    logger.warn("Redis unavailable — running without cache (start Redis to enable caching)");
  }
}

export function isCacheAvailable(): boolean {
  return redis.isReady;
}

export async function cacheGet<T>(key: string): Promise<T | null> {
```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

if (!redis.isReady) return null;

try {
  const value = await redis.get(`${PREFIX}${key}`);
  return value ? (JSON.parse(value) as T) : null;
} catch (err) {
  logger.warn({ err, key }, "Cache get failed");
  return null;
}
}

export async function cacheSet(key: string, value: unknown, ttlSeconds?: number): Promise<void>
{
  if (!redis.isReady) return;

  try {
    const serialized = JSON.stringify(value);
    if (ttlSeconds) await redis.setEx(`${PREFIX}${key}`, ttlSeconds, serialized);
    else await redis.set(`${PREFIX}${key}`, serialized);
  } catch (err) {
    logger.warn({ err, key }, "Cache set failed");
  }
}

export async function cacheSetIfNotExists(
  key: string, value: unknown, ttlSeconds: number,
): Promise<boolean> {
  if (!redis.isReady) return false;

  try {
    const result = await redis.set(`${PREFIX}${key}`, JSON.stringify(value), {
      NX: true, EX: ttlSeconds,
    });
    return result === "OK";
  } catch (err) {
    logger.warn({ err, key }, "Cache set-if-not-exists failed");
    return false;
  }
}

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

```

export async function cacheDelByPrefix(prefix: string): Promise<number> {
  if (!redis.isReady) return 0;
  let deleted = 0;
  try {
    const batch: string[] = [];
    for await (const key of redis.scanIterator({ MATCH: `${PREFIX}${prefix}*`, COUNT: 200 })) {
      batch.push(key as string);
      if (batch.length >= 200) { deleted += await redis.del(batch); batch.length = 0; }
    }
    if (batch.length > 0) deleted += await redis.del(batch);
  } catch (err) {
    logger.warn({ err, prefix }, "Cache purge failed");
  }
  return deleted;
}

export const CacheKeys = {
  player: (steamId: string) => `player:${steamId}`,
  playerRefreshLock: (steamId: string) => `player-refresh-lock:${steamId}`,
  leaderboard: (region: string, page: number) => `leaderboard:${region}:${page}`,
  leaderboardSnap: (region: string) => `leaderboard-snapshot:${region}:v5`,
  leaderboardLock: (region: string) => `leaderboard-refresh-lock:${region}`,
  heroes: () => `heroes:all`,
  items: () => `items:all`,
};

```

deadlockApi.ts

```

const MAX_RETRIES = 2;

async function deadlockRequest<T>(
  fn: () => Promise<AxiosResponse<T>>,
  endpoint: string,
): Promise<T> {
  for (let attempt = 0; attempt <= MAX_RETRIES; attempt++) {
    try {
      const res = await fn();
      return res.data;
    } catch (err: any) {

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		90

```

const status = err.response?.status as number | undefined;
const message = (err.response?.data?.message ?? err.message) as string;
if ((status === 429 || status === 503) && attempt < MAX_RETRIES) {
const retryAfter = err.response?.headers?.["retry-after"];
const delay = retryAfter
? Math.min(parseInt(retryAfter, 10) * 1000, 5_000)
: 1_000 * (attempt + 1);
logger.warn({ endpoint, status, attempt, delay }, "Deadlock API rate limited, retrying");
await new Promise((r) => setTimeout(r, delay));
continue;
}
if (err.code === "ECONNABORTED" || err.code === "ETIMEDOUT") {
throw new ApiError("Request timeout", 408, endpoint);
}
if (status === 404) throw new ApiError("Not found", 404, endpoint);
throw new ApiError(message, status ?? 500, endpoint);
}
}
throw new ApiError("Max retries exceeded", 429, endpoint);
}
export async function getBatchPlayerHeroStats(
accountIds: number[],
): Promise<DeadlockPlayerHeroStat[]> {
if (accountIds.length === 0) return [];
return deadlockRequest(
() => apiClient.get<DeadlockPlayerHeroStat[]>("/v1/players/hero-stats", {
params: { account_ids: accountIds.join(",") },
timeout: 30_000,
}),
"/v1/players/hero-stats (batch)",
);
}

```

leaderboardSnapshot.ts

```

const SNAPSHOT_TTL_SECONDS = 604_800;
const SNAPSHOT_FRESHNESS_SECONDS = 172_800;

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```

const LOCK_TTL_SECONDS = 1_800;

async function buildQuickSnapshot(region: DeadlockRegion): Promise<LeaderboardSnapshot> {
  const raw = await getLeaderboard(region);
  const entries = raw.map((entry) => ({
    ...entry,
    resolvedAccountId: entry.possible_account_ids[0] ?? null,
    confident: entry.possible_account_ids.length === 1,
  }));
  return { fetchedAt: new Date().toISOString(), resolved: false, entries };
}

async function buildFullSnapshot(region: DeadlockRegion): Promise<LeaderboardSnapshot> {
  const raw = await getLeaderboard(region);
  const resolvedMap = await resolveAccountIds(raw).catch(() => new Map());
  const entries = raw.map((entry, idx) => {
    const r = resolvedMap.get(idx);
    return {
      ...entry,
      resolvedAccountId: r?.accountId ?? entry.possible_account_ids[0] ?? null,
      confident: r?.confident ?? entry.possible_account_ids.length === 1,
    };
  });
  return { fetchedAt: new Date().toISOString(), resolved: true, entries };
}

function scheduleFullResolution(region: DeadlockRegion, snapKey: string, lockKey: string): void {
  buildFullSnapshot(region)
    .then(async (snap) => {
      await cacheSet(snapKey, snap, SNAPSHOT_TTL_SECONDS);
      await cacheDelByPrefix(`leaderboard:${region}`);
    })
    .catch((err) => logger.warn({ region, err }, "Background leaderboard resolution failed"))
    .finally(() => cacheDel(lockKey));
}

export async function getLeaderboardSnapshot(region: DeadlockRegion):
Promise<LeaderboardSnapshot> {
  const snapKey = CacheKeys.leaderboardSnap(region);

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

const lockKey = CacheKeys.leaderboardLock(region);
if (!isCacheAvailable()) return buildQuickSnapshot(region);
const cached = await cacheGet<LeaderboardSnapshot>(snapKey);
if (cached && !isSnapshotStale(cached)) {
  if (!cached.resolved) {
    const lockAcquired = await cacheSetIfNotExists(lockKey, "1", LOCK_TTL_SECONDS);
    if (lockAcquired) scheduleFullResolution(region, snapKey, lockKey);
  }
  return cached;
}
const lockAcquired = await cacheSetIfNotExists(lockKey, "1", LOCK_TTL_SECONDS);
if (!lockAcquired) return cached ?? buildQuickSnapshot(region);
const quickSnap = await buildQuickSnapshot(region);
await cacheSet(snapKey, quickSnap, SNAPSHOT_TTL_SECONDS);
scheduleFullResolution(region, snapKey, lockKey);
return quickSnap;
}

```

leaderboardSnapshot.ts

```

const BANNED_STEMS = [
  // ...повний перелік у лістингу файла
];

function normalizeToken(token: string): string {
  return token.toLowerCase().replace(/[^a-zA-яёіієг]/gi, "");
}

export function isProfane(name: string): boolean {
  if (!name) return false;
  return name.split(/\s+/).some((token) => {
    const n = normalizeToken(token);
    return n.length >= 3 && BANNED_STEMS.some((s) => n.includes(s));
  });
}

export function safePlayerName(name: string | null | undefined, id?: string | number | null): string {
  if (name && !isProfane(name)) return name;
  return id != null ? `Player ${id}` : "Player";
}

```

		Балюк В.В.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Кушнір Н.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		93