Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Медіасистеми та технології
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

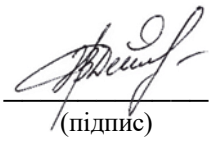
Дослідження процесу робзробки міні додатків Telegram
з використанням web-технології
(тема)

Виконав:
здобувач 2 року навчання
групи КТСВПВм-24-1

 Костянтин ДЕМЧЕНКО
(власне ім'я, прізвище)

Спеціальність 186 Видавництво та поліграфія
(код і повна назва спеціальності)
Тип програми Освітньо-професійна
Освітня програма
Комп'ютерні технології та системи
видавничо-поліграфічних виробництв

Керівник  проф. Нонна КУЛІШОВА
(посада, власне ім'я, прізвище)

Допускається до захисту
Завідувач кафедри МСТ  Жанна ДЕЙНЕКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Медіасистеми та технології
Рівень вищої освіти другий (магістерський)
Спеціальність 186 Видавництво та поліграфія
Тип програми Освітньо-професійна
Освітня програма Комп'ютерні технології
та системи видавничо-поліграфічних виробництв
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри МСТ _____
(підпис)
« 03 » листопада 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві *Демченка Костянтину Сергійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження процесу робзробки міні додатків Telegram з використанням web-технології*

затверджена наказом по університету від *03 листопада 2025 р. № 988 Ст*

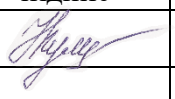
2. Термін подання здобувачем роботи до екзаменаційної комісії *18 грудня 2025 р.*

3. Вихідні дані до роботи
Офіційна документація Telegram Bot API та Mini Apps; документація веб-фреймворків (Next.js, React); принципи побудови архітектури високонавантажених веб-систем; сучасні підходи до реалізації GameFi-проєктів.

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз літератури та предметної області (еволюція месенджерів, феномен Mini Apps); Аналіз існуючих аналогів (GameFi проєкти в Telegram) та їх недоліків; Обґрунтування вибору технологічного стеку (Next.js, Node.js, PostgreSQL); Проєктування архітектури системи: клієнт-серверна взаємодія, безпека (валідація initData), схема бази даних; Розробка дизайн-концепції та UI/UX інтерфейсу ігрового додатку (адаптивність, темна/світла теми); Програмна реалізація основних модулів гри (механіка клікера, магазин, інвентар скінів); Експериментальне дослідження (тестування продуктивності, навантажувальне тестування); Економічна частина (обґрунтування ефективності проєкту, ROI); Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій
Актуальність, мета, завдання та об'єкт дослідження; Класифікація та аналіз аналогів (Notcoin, Hamster Kombat тощо); Архітектура розробленої системи (схема взаємодіїкомпонентів); Схема бази даних (ER-діаграма); Інтерфейс розробленого мобільного додатку (екрани гри); Алгоритми роботи ключових модулів (блок-схеми); Результати експериментального дослідження (графіки продуктивності); Показники економічної ефективності; Висновки.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	проф. Кулішова Н.Є.		09.12.25
Економічна частина	доц. Потій О.О.		17.12.25

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз літератури та існуючих аналогів	08.11	виконано
2	Формулювання вимог та гіпотез дослідження	14.11	виконано
3	Проєктування архітектури системи та бази даних	20.11	виконано
4	Програмна реалізація клієнтської та серверної части	30.11	виконано
5	Проведення експериментального дослідження	05.12	виконано
6	Економічна частина	10.12	виконано
7	Оформлення пояснювальної записки	15.12	виконано
8	Оформлення графічного матеріалу та презентації	17.12	виконано
9	Захист кваліфікаційної роботи	19.12	виконано

Дата видачі завдання 03 листопада 2025 р.

Здобувач 
(підпис)

Керівник роботи  проф. Нонна КУЛІШОВА
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить 52 с., 4 табл., 6 рис., 1 дод., 28 джерел.

ТЕЛГРАМ МІНІ ДОДАТКИ, ВЕБ-ТЕХНОЛОГІЇ, КЛІКЕР, РОЗРОБКА ІГРОВИХ ДОДАТКІВ, NEXT.JS, NODE.JS, UI/UX.

Метою роботи є дослідження особливостей технології Telegram Mini Apps та розробка ігрового веб-додатку (клікера з механікою колекціонування скінів) із використанням сучасного стеку веб-технологій, що забезпечує високу продуктивність та безшовну інтеграцію з екосистемою месенджера.

Об’єктом дослідження є процес створення інтерактивних веб-застосунків у середовищі месенджера Telegram. У роботі проведено аналіз існуючих аналогів на ринку GameFi, обґрунтовано вибір архітектурного паттерну клієнт-сервер та реалізовано прототип гри. Основну увагу приділено питанням оптимізації рендерингу (60 FPS), безпеки передачі даних (initData) та проєктуванню інтуїтивно зрозумілого інтерфейсу для мобільних пристроїв. Результати роботи підтверджують ефективність використання Telegram Mini Apps для створення віральних ігрових продуктів.

ABSTRACT

Explanatory note of qualified work has 52 p., 5 tabl., 6 fig., 1 app, 26 sources.

TELEGRAM MINI APPS, GAMEFI, WEB TECHNOLOGIES, CLICKER, GAME DEVELOPMENT, NEXT.JS, NODE.JS, UI/UX.

The purpose of the work is to research the technology of developing Telegram Mini Apps and to create a game-oriented software product (a clicker game with CS skins theme) using modern web technologies, ensuring high performance and seamless integration with the messenger ecosystem.

The object of the study is the process of creating interactive web applications within the Telegram messenger environment. The paper analyzes existing analogues in the GameFi market, substantiates the choice of the client-server architecture pattern, and implements a game prototype. Special attention is paid to rendering optimization (60 FPS), data transmission security (initData), and designing an intuitive interface for mobile devices. The results of the work confirm the effectiveness of using Telegram Mini Apps for creating viral game products.

ЗМІСТ

С.

СКОРОЧЕЕННЯ ТА УМОВНІ ПОЗНАКИ

ACID (Atomicity, Consistency, Isolation, Durability) – набір властивостей транзакцій бази даних (атомарність, узгодженість, ізольованість, довговічність), що гарантують надійність та цілісність даних при виконанні фінансових операцій.

API (Application Programming Interface) – набір визначень підпрограм, протоколів та інструментів для взаємодії між різними програмними компонентами (наприклад, між клієнтом та сервером).

CMS (Content Management System) – система управління контентом; у даній роботі роль «Headless CMS» виконує сервіс Notion для адміністрування товарів.

CPA (Cost Per Action) – модель інтернет-реклами, де рекламодавець платить за здійснення користувачем певної цільової дії (наприклад, підписки на канал).

CSS (Cascading Style Sheets) – формальна мова опису зовнішнього вигляду документа, написаного мовою розмітки; використовується для стилізації інтерфейсу.

DOM (Document Object Model) – об'єктна модель документа; програмний інтерфейс для HTML-документів, що дозволяє скриптам змінювати вміст, структуру та стиль веб-сторінки.

FPS (Frames Per Second) – кадрова частота; кількість змін кадрів за одиницю часу, що характеризує плавність відтворення анімації в інтерфейсі.

HMAC (Hash-based Message Authentication Code) – код автентифікації повідомлень, що використовує хеш-функції; застосовується для валідації даних ініціалізації (initData) від Telegram.

HTML (HyperText Markup Language) – стандартизована мова розмітки документів для перегляду веб-сторінок у браузері.

HTTP/HTTPS (HyperText Transfer Protocol / Secure) – протокол передачі даних у комп'ютерних мережах; використовується для обміну інформацією між клієнтом і сервером.

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на синтаксисі об'єктів JavaScript; використовується для зберігання конфігурацій та передачі даних API.

MVP (Minimum Viable Product) – мінімально життєздатний продукт; версія продукту з мінімальним набором функцій, достатнім для задоволення перших споживачів.

PWA (Progressive Web App) – технологія у веб-розробці, що дозволяє створювати веб-сайти, які візуально і функціонально нагадують нативні мобільні додатки.

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого додатка в мережі; використовується при побудові API.

ROI (Return on Investment) – коефіцієнт рентабельності інвестицій; фінансовий показник, що ілюструє рівень прибутковості чи збитковості проєкту.

SPA (Single Page Application) – односторінковий веб-застосунок, який завантажує єдину HTML-сторінку та динамічно оновлює її вміст без перезавантаження, забезпечуючи високу швидкість роботи.

SSR (Server-Side Rendering) – рендеринг на стороні сервера; технологія, при якій HTML-сторінка генерується на сервері та відправляється клієнту вже готовою для відображення.

TMA (Telegram Mini App) – веб-застосунок, що запускається всередині месенджера Telegram і має доступ до його нативного API.

UI (User Interface) – інтерфейс користувача; сукупність засобів, за допомогою яких користувач взаємодіє з програмою (кнопки, меню, екрани).

UX (User Experience) – досвід користувача; сукупність вражень, які отримує користувач при взаємодії з продуктом (зручність, зрозумілість, швидкість реакції).

VPS (Virtual Private Server) – послуга надання в оренду віртуального виділеного сервера для розміщення програмного забезпечення.

WebSocket (WSS) – протокол зв'язку, що забезпечує повнодуплексний канал обміну даними через одне TCP-з'єднання; використовується для обробки подій реального часу (кліків).

ВСТУП

У сучасному цифровому середовищі месенджери трансформуються із засобів комунікації у повноцінні платформи для запуску застосунків. Окреме місце серед таких рішень посідають міні-додатки Telegram, або Telegram Mini Apps, які дозволяють інтегрувати інтерактивні веб-сервіси безпосередньо в інтерфейс чату. Це відкриває нові можливості для індустрії розваг, зокрема для жанру GameFi, де швидкість доступу та соціальна взаємодія є критичними факторами успіху.

Актуальність дослідження зумовлена стрімким зростанням популярності міні-програм у екосистемі Telegram та запитом користувачів на так званий легкий геймінг без необхідності встановлення окремих інсталяційних файлів форматів APK чи IPA. Клікер-механіка моделі «натискай та заробляй» у поєднанні з колекціонуванням віртуальних активів, а саме скінів з популярної гри Counter-Strike, є ефективною моделлю залучення аудиторії. Вона демонструє високі показники утримання користувачів завдяки коротким ігровим сесіям та миттєвому зворотному зв'язку. Використання сучасного веб-стеку на базі React та Next.js дозволяє забезпечити нативну продуктивність таких рішень.

Мета роботи полягає у дослідженні технології розробки міні-додатків Telegram та створенні програмного продукту ігрового спрямування, а саме клікер-гри з тематикою CS-скінів, із використанням сучасних веб-технологій. Для досягнення мети поставлено такі завдання:

– проаналізувати сучасний стан технології Telegram Mini Apps, існуючі аналоги та архітектурні підходи до їх реалізації;

– обґрунтувати вибір технологічного стеку, що включає Next.js, Nest.js, MariaDB, WebSocket та Docker, для забезпечення продуктивності та масштабованості системи;

– спроєктувати архітектуру клієнт-серверної взаємодії з урахуванням специфіки авторизації через параметри ініціалізації Telegram та безпеки даних;

– розробити та програмно реалізувати прототип гри, що включає механіки клікера, систему інвентарю, магазин скінів та реферальну систему;

– провести тестування розробленого рішення, оцінити його ефективність та економічну доцільність впровадження.

Об’єкт дослідження – процес створення інтерактивних веб-застосунків у середовищі месенджера Telegram.

Предмет дослідження – методи та засоби розробки Telegram Mini Apps ігрового спрямування, зокрема архітектурні патерни, методи оптимізації UI/UX та механізми інтеграції з API Telegram.

У роботі використано методи системного аналізу для дослідження предметної області, об’єктно-орієнтованого проєктування для побудови архітектури, прототипування для розробки інтерфейсу та експериментального тестування для перевірки продуктивності.

Наукова новизна одержаних результатів полягає у вдосконаленні підходів до інтеграції веб-технологій у середовище месенджерів для створення GameFi продуктів. Зокрема, запропоновано архітектурне рішення для безпечної синхронізації ігрового стану між клієнтом і сервером в умовах нестабільного з'єднання з використанням протоколу WebSocket.

Розроблено повнофункціональний прототип гри-клікера, який може бути використаний як основа для комерційного продукту або як платформа для подальших досліджень у сфері мобільного веб-геймінгу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ РОЗРОБКИ TELEGRAM MINI APPS

1.1 Еволюція месенджерних інтеграцій та концепція Telegram Mini Apps

Історія автоматизованих систем взаємодії з користувачем у месенджерах пройшла значний шлях еволюції: від простих текстових ботів до повноцінних вбудованих веб-платформ. Спочатку інструментарій, що надавався розробникам, обмежувався текстовими командами та простими меню (перші версії Telegram Bot API 1.0–2.0), що дозволяло створювати лише базові інформаційні сервіси [1]. Згодом функціонал розширився: з'явилися інлайн-кнопки та розширені медіа-можливості, проте інтерфейс користувача все ще залишався жорстко прив'язаним до структури чату.

Революційним кроком у розвитку екосистеми стала поява технології Telegram Mini Apps (Web Apps), яку було презентовано в оновленні Bot API 6.0 [2]. Це гібридна технологія, що дозволяє відкривати веб-сторінки безпосередньо всередині клієнта Telegram через вбудований системний браузер (WebView). На відміну від класичних ботів, Web Apps надають розробнику повний контроль над інтерфейсом через DOM-дерево, що дозволяє реалізовувати сценарії будь-якої складності: від інтернет-магазинів до інтерактивних ігор з плавною анімацією 60 FPS [3].

З точки зору архітектури інформаційних систем, Telegram Mini Apps виконують роль «миттєвого фронтенду». Вони не потребують завантаження з App Store чи Google Play, що радикально знижує поріг входу для користувача та дозволяє обходити обмеження класичних магазинів застосунків. Ключовою особливістю платформи є безшовна інтеграція з екосистемою Telegram:

– застосунок автоматично отримує дані користувача через об'єкт initData, що усуває потребу в створенні окремих екранів логіну або реєстрації та дозволяє миттєво ідентифікувати користувача [4];

– можливість зчитувати системні кольори Telegram (tgWebAppThemeParams) дозволяє адаптувати дизайн під налаштування користувача (світла/темна тема) [5];

– кросплатформність: один і той самий веб-клієнт коректно працює на iOS, Android, macOS та Windows без необхідності розробки окремих версій;

– доступ до апаратних функцій: використання HapticFeedback (вібровідгук). Використання тактильного зворотного зв'язку при натисканні кнопок є критично важливим для покращення UX у іграх жанру «клікер» [6].

Таким чином, Telegram Mini Apps являють собою еволюційний етап розвитку мобільного вебу, який поєднує доступність веб-технологій з глибиною інтеграції нативних додатків.

1.2 Аналіз ринку GameFi-проєктів та існуючих аналогів

На момент проведення дослідження ніша ігрових міні-додатків у Telegram переживає період експоненціального зростання. Цей тренд, що отримав назву «Tap-to-Earn», базується на синергії віральних ігрових механік та можливостей блокчейн-мережі The Open Network TON. Згідно з аналітичним звітом Earlybird за 2025 рік, інтеграція Web3-технологій у месенджери стала основним драйвером масового залучення користувачів у криптоіндустрію, а кількість активних гаманців у мережі TON зросла в рази саме завдяки ігровим ботам [7].

Феномен популярності цих додатків пояснюється психологічним ефектом «миттєвої винагороди» та низьким порогом входу: користувачеві не потрібно встановлювати стороннє програмне забезпечення або проходити складну процедуру реєстрації на старті.

Для детального розуміння предметної області було проведено порівняльний аналіз ключових гравців ринку.

1. Notcoin [8] – проєкт, що став першопрохідцем жанру та довів життєздатність концепції Telegram Mini Apps для високо навантажених систем.

Механіка базується на максимально спрощеній дії – натисканні на зображення монети для майнінгу віртуальної валюти.

Технічні особливості: розробники Notcoin продемонстрували ефективність використання протоколу WebSocket для передачі даних про кліки в реальному часі, що дозволило системі витримувати пікові навантаження у мільйони одночасних користувачів (DAU).

Недоліки: з точки зору геймдизайну, візуальна складова є мінімалістичною, а ігровий процес – лінійним та одноманітним, що з часом призводить до зниження залученості після завершення фази майнінгу.

2. Hamster Kombat [9]. Цей проєкт є еволюційним розвитком жанру, який трансформував простий клікер у симулятор CEO криптовалютної біржі.

Механіка: основний акцент зміщено з активного клікання на стратегічне планування пасивного доходу. Гра використовує складну систему карток, рівнів прокачування та щоденних комбо-завдань, що стимулює користувачів повертатися у додаток кожні кілька годин.

Особливості: використання елементів соціальної інженерії та вірального маркетингу (обов'язкова підписка на канали для відкриття карток) дозволило проєкту зібрати рекордну аудиторію за короткий час.

Недоліки: економічна модель повністю залежить від спекулятивних очікувань лістингу токена. Затримки з виплатами або низька вартість токена можуть миттєво знецінити витрачений час гравців.

3. Blum [10] – гібридне рішення, що позиціонується розробниками не просто як гра, а як повноцінна децентралізована біржа з елементами гейміфікації.

Механіка: користувачі заробляють бали (Blum Points) через виконання завдань та міні-гру в жанрі «Drop game», які згодом конвертуються у реальні торгові привілеї.

Технічні особливості: інтерфейс додатку реалізовано як SPA (Single Page Application) з високою чутливістю, що забезпечує досвід користування, наближений до нативних мобільних додатків.

Проблематика існуючих рішень та обґрунтування власного підходу Попри комерційний успіх згаданих аналогів, аналіз виявив системну проблему сегмента: інфляція очікувань. Більшість проєктів пропонують користувачеві винагороду у вигляді абстрактних токенів, реальна вартість яких невідома до моменту лістингу на біржі. Це створює ризик розчарування аудиторії та відтоку користувачів (Churn Rate). Крім того, візуальний ряд більшості клікерів є однотипним і швидко набридає.

У розроблюваному проєкті пропонується вирішити цю проблему шляхом зміни парадигми винагороди. Замість відкладеної в часі виплати токенів, впроваджується інтеграція механіки магазину скінів з гри Counter-Strike 2, що включає такі аспекти:

– цінність активів: скіни CS мають сформовану ринкову вартість на платформі Steam і є ліквідним товаром;

– мотивація: можливість придбати конкретний візуально привабливий предмет за зароблену внутрішню валюту створює прозору та зрозумілу ціль для гравця;

– технічна реалізація: використання Steam Trade Link для виведення предметів дозволяє реалізувати винагороду без необхідності підключення криптогаманців, що знижує поріг входу для аудиторії, не знайомої з Web3.

Такий підхід дозволяє перетворити абстрактні «кліки» на реальну цінність, що значно підвищує показники утримання аудиторії (Retention) та створює стійку базу для монетизації через CPA-модель.

1.3 Обґрунтування вибору технологічного стеку

Для реалізації високонавантаженої ігрової системи з мікросервісною архітектурою було проведено детальний аналіз сучасних інструментів веб-розробки. Вибір технологічного стеку базувався на комплексному підході, що враховує критерії продуктивності (Performance), можливості горизонтального

масштабування (Scalability), швидкості розробки (Time-to-Market) та зручності підтримки коду (Maintainability) у довгостроковій перспективі.

Розробка клієнтської частини (Frontend) базується на мета-фреймворку Next.js [11], побудованому на бібліотеці React. Вибір саме цього інструменту, а не класичного React (CRA) чи Vue.js, зумовлений специфікою роботи Telegram Web Apps. Оскільки додаток працює всередині мобільного WebView, критичним показником якості є час першого відмальовування контенту (First Contentful Paint). Next.js вирішує цю проблему за допомогою технології Server-Side Rendering (SSR) та Static Site Generation (SSG), що дозволяє генерувати HTML-розмітку на сервері та відправляти клієнту вже готовий інтерфейс, мінімізуючи час "білого екрану" під час ініціалізації. Крім того, вбудований механізм оптимізації зображень (next/image) автоматично конвертує графічні асети (скіни, іконки) у сучасний формат WebP та адаптує їх розмір під роздільну здатність пристрою користувача, що суттєво зменшує обсяг трафіку.

Для забезпечення надійності кодової бази та мінімізації помилок на етапі розробки використовується мова програмування TypeScript [12]. У проєктах, пов'язаних з фінансовими операціями (навіть ігровою валютою) та складною бізнес-логікою, сувора статична типізація є необхідністю. TypeScript дозволяє створити єдині інтерфейси даних (Data Transfer Objects – DTO), які використовуються як на клієнті, так і на сервері, гарантуючи консистентність даних при обміні повідомленнями через API. Це особливо важливо при роботі зі структурою initData від Telegram, яка містить чутливі дані користувача.

Візуалізація інтерфейсу та адаптивність реалізовані за допомогою Tailwind CSS [13]. Цей utility-first CSS фреймворк дозволяє створювати складні інтерфейси без написання окремих файлів стилів, використовуючи заздалегідь визначені класи безпосередньо у розмітці (JSX). Важливою перевагою Tailwind у контексті Telegram Mini Apps є використання JIT-компілятора (Just-In-Time), який під час збірки проєкту генерує CSS-файл, що містить лише ті стилі, які фактично використовуються в додатку. Це забезпечує мінімальний розмір бандлу стилів, що позитивно впливає на швидкість завантаження додатку в

умовах мобільного інтернету. Крім того, Tailwind значно спрощує реалізацію темної та світлої тем, дозволяючи використовувати модифікатор dark: для адаптації під системні налаштування Telegram.

Серверна частина (Backend) реалізована на базі фреймворку Nest.js [14], що працює у середовищі виконання Node.js. На відміну від мінімалістичного Express.js, Nest.js пропонує чітку архітектуру "з коробки", що базується на модулях, контролерах та провайдерах. Ключовою перевагою є активне використання патерну Dependency Injection (DI) та декораторів, що робить код декларативним, тестованим та слабкозв'язним. Для гри жанру «клікер», де основне навантаження створюють тисячі дрібних запитів (кліків) в секунду, критично важливою є підтримка протоколу WebSocket. Nest.js має вбудований модуль Gateway, який дозволяє легко реалізувати WebSocket-сервер для двостороннього обміну даними в реальному часі, забезпечуючи миттєву синхронізацію балансу користувача без необхідності постійних HTTP-запитів (polling).

У якості основної системи управління базами даних обрано MariaDB [15]. Це високопродуктивна реляційна СУБД, яка є бінарно сумісним відгалуженням MySQL. Вибір реляційної моделі (SQL) замість NoSQL (наприклад, MongoDB) обумовлений структурою даних предметної області: наявністю чітких зв'язків між сутностями (один користувач має багато предметів в інвентарі, одне замовлення прив'язане до одного користувача). MariaDB забезпечує відповідність принципам ACID (Atomicity, Consistency, Isolation, Durability), що є обов'язковою вимогою для реалізації транзакційних операцій. Наприклад, операція купівлі скіна вимагає одночасного списання балансу та додавання запису в інвентар – якщо одна з дій не вдасться, транзакція має бути повністю відкочена, щоб уникнути втрати коштів або дюпу предметів.

Для управління контентом та адміністрування магазину застосовано інноваційний архітектурний підхід із використанням сервісу Notion [16] у якості Headless CMS. Замість витрат ресурсів на розробку власної адміністративної панелі (Frontend Admin UI), використовується гнучкість баз даних Notion. Менеджери проєкту додають нові скіни, редагують ціни та

описи у звичному інтерфейсі Notion. Інтеграція реалізована через Notion API: спеціальний сервіс на бекенді періодично синхронізує дані з Notion у виробничу базу даних MariaDB. Це дозволяє поєднати зручність редагування (Low-code) з високою швидкодією читання даних для кінцевих користувачів, оскільки клієнтський додаток звертається до швидкої MariaDB, а не до лімітованого API Notion.

Для забезпечення стабільності розгортання, ізоляції середовища виконання та спрощення процесу CI/CD (Continuous Integration / Continuous Delivery) всі компоненти системи (клієнт, сервер, база даних) контейнеризовано за допомогою платформи Docker [17]. Використання Docker Compose дозволяє описати інфраструктуру проєкту у вигляді коду (Infrastructure as Code), що гарантує ідентичність оточення на локальних машинах розробників та на продуктовому сервері, усуваючи класичну проблему «it works on my machine».

1.4 Обґрунтування архітектури Mini App у порівнянні з PWA

При проєктуванні архітектури системи було проведено детальний порівняльний аналіз двох основних підходів до розробки кросплатформних мобільних рішень: класичних прогресивних веб-застосунків (Progressive Web Apps – PWA) та спеціалізованих міні-додатків Telegram (Mini Apps). Хоча обидві технології базуються на ідентичному веб-стеку (HTML/CSS/JavaScript), вони пропонують принципово різні моделі взаємодії з користувачем, що безпосередньо впливає на конверсію та утримання аудиторії.

Критичним фактором вибору архітектури став аналіз так званої «воронки залучення» (User Acquisition Funnel). У випадку з класичними PWA, для отримання повноекранного досвіду користувач змушений виконувати низку неочевидних дій, зокрема вручну додавати сайт на головний екран пристрою через системне меню браузера. Статистика свідчить, що на цьому етапі втрачається значна частина аудиторії, особливо на пристроях під управлінням

iOS, де цей функціонал є менш інтуїтивним [18]. Натомість архітектура Telegram Mini App усуває цей бар'єр: додаток відкривається миттєво всередині чату за натисканням однієї кнопки, не вимагаючи виходу зі звичного середовища месенджера, що забезпечує максимально можливу конверсію з перегляду реклами у першу ігрову сесію. Іншим вагомим аргументом на користь інтегрованого рішення є питання ідентифікації користувача та безпеки сесії. Розробка PWA неминуче стикається з проблемою відсутності доступу до унікального ідентифікатора користувача Telegram, що змушує впроваджувати класичні методи авторизації через електронну пошту або SMS. Це не лише створює додаткові витрати на верифікацію, але й підвищує ризик відтоку користувачів, які не бажають проходити процедуру реєстрації. У середовищі Mini Apps ця проблема вирішується на рівні платформи: механізм передачі об'єкта initData дозволяє реалізувати так звану «тиху авторизацію», де бекенд автоматично валідує криптографічний підпис Telegram, гарантуючи автентичність користувача без жодних додаткових дій з його боку.

Окремої уваги заслуговує інтеграція з соціальним графом користувача, що є критично важливим для ігор із віральними механіками росту. Використання PWA призводить до розриву контексту (Context Switching), коли для запрошення друга користувач змушений згортати браузер, копіювати посилання та вручну відправляти його в месенджері. Telegram Mini Apps надають нативні методи для шерингу та взаємодії з контактами, дозволяючи відправити запрошення в один клік безпосередньо з інтерфейсу гри [19]. Крім того, доступ до бібліотеки Telegram WebApp SDK дозволяє використовувати апаратні можливості пристрою, зокрема тактильний відгук (Haptic Feedback), який часто заблокований або обмежений у мобільних браузерах, але є необхідним для створення якісного ігрового досвіду в жанрі клікерів.

Таким чином, архітектура Telegram Mini App визначена як найбільш доцільна для реалізації казуальної гри, оскільки вона поєднує гнучкість веб-розробки з UX-перевагами нативних додатків, забезпечуючи найнижчий поріг входу та максимальне використання екосистеми для утримання користувачів.

2 ФОРМУЛЮВАННЯ ГІПОТЕЗИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

2.1 Обґрунтування доцільності розробки

Аналіз ринку, проведений у першому розділі, продемонстрував, що класичні клікери швидко втрачають аудиторію через відсутність довгострокової цінності для користувача. Водночас існує стійкий попит з боку рекламодавців на залучення активної аудиторії у Telegram-канали.

Доцільність розробки полягає у створенні платформи, яка вирішує завдання двох сторін:

– для користувача: надає можливість отримати бажані цифрові товари як скіни CS безкоштовно, інвестуючи лише свій час та соціальну активність;

– для бізнесу (власника): генерує високоякісний трафік за моделлю CPA (Cost Per Action). Утримання користувача досягається через гейміфікацію процесу накопичення валюти, а ріст аудиторії – через віральні механіки (реферальна система).

Утримання користувача досягається через гейміфікацію процесу накопичення валюти, а зростання аудиторії – через використання вірaльних механік реферальної системи. З технічної точки зору, система повинна витримувати пікові навантаження та забезпечувати миттєву доставку контенту, що вимагає застосування гібридного архітектурного підходу.

2.2 Формування вимог до системи

Система проєктується як високо навантажений веб-сервіс із мікросервісними елементами.

1. Функціональні вимоги. Основний функціонал системи розподілено на чотири ключові модулі. CPA-модуль відповідає за монетизацію уваги

користувача. Він повинен забезпечувати автоматичну перевірку підписок на Telegram-канали партнерів та нарахування винагороди за виконання цільових дій. Віральна механіка реалізується через реферальну систему, яка включає генерацію унікальних посилань, трекінг запрошених користувачів та нарахування бонусів у вигляді відсотка від заробітку реферала.

Економічна система будується навколо магазину скінів. Вона має забезпечувати відображення каталогу товарів із актуальними цінами, обробку замовлень, валідація балансу, створення заявки, списання коштів, та інтеграцію Steam Trade Link для доставки предмета користувачу. Адміністрування контенту реалізується за гібридною схемою Hybrid CMS. Використання Notion дозволяє менеджерам зручно додавати та редагувати товари, а фонова синхронізація даних з Notion у виробничу базу даних MariaDB гарантує високу швидкодію для кінцевого користувача.

2. Нефункціональні вимоги. Для забезпечення якісного користувацького досвіду (UX) система повинна відповідати суворим нефункціональним вимогам. Критичним показником є швидкодія: час відповіді API на запит списку товарів не повинен перевищувати 100 мс. Це досягається завдяки тому, що клієнтська частина звертається до оптимізованої бази даних MariaDB, а не безпосередньо до повільного API Notion. Архітектура повинна забезпечувати масштабованість, дозволяючи горизонтальне розширення потужностей при різкому напливі трафіку. Також важливою є надійність даних, що передбачає захист від «накрутки» кліків та фальсифікації рефералів.

2.3 Критерії оцінювання ефективності

До технічних показників віднесено мережеву затримку– різницю у часі відгуку при прямому запиті до Notion API порівняно із запитом до синхронізованої бази даних, а також стабільність частоти кадрів під час активної взаємодії з інтерфейсом.

Бізнес-показники включають рівень утримання користувачів, що демонструє ефективність механіки магазину, та віральний коефіцієнт, який відображає кількість нових користувачів, залучених одним активним гравцем.

2.4 Формулювання гіпотез дослідження

На основі аналізу бізнес-моделі та технічних обмежень висуваються наступні гіпотези.

Гіпотеза 1. Побудова ігрової економіки навколо реальних цифрових активів (скіни CS), що виступають винагородою за виконання завдань, забезпечує вищий рівень утримання аудиторії та органічного росту порівняно з моделями, що базуються виключно на спекулятивному очікуванні лістингу токенів. Матеріальна цінність активів виступає ключовим фактором мотивації.

Гіпотеза 2. Реалізація гібридної архітектури управління контентом, де Notion використовується як джерело даних, а MariaDB як шар кешування, дозволяє поєднати високу гнучкість No-Code інструментів із продуктивністю традиційних баз даних. Такий підхід нівелює технічні обмеження зовнішніх API (обмеження кількості запитів, затримки) та усуває необхідність витрат на розробку кастомного адміністративного інтерфейсу, прискорюючи вихід продукту на ринок.

3 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Загальна архітектура та взаємодія компонентів

Для реалізації ігрової платформи обрано мікросервісний підхід із чітким розділенням відповідальності між компонентами системи, що дозволяє забезпечити гнучкість розробки та легкість масштабування. Архітектура розробленого програмного комплексу складається з трьох основних рівнів: клієнтського, серверного та рівня зберігання даних. Взаємодія між цими рівнями реалізована за допомогою гібридної моделі зв'язку, яка використовує протокол HTTP для транзакційних запитів та WebSocket для обробки подій реального часу, що є критичним для забезпечення миттєвого зворотного зв'язку в іграх жанру «клікер».

Загальна схема архітектури системи та потоків даних зображена на рисунку (рис. 3.1).

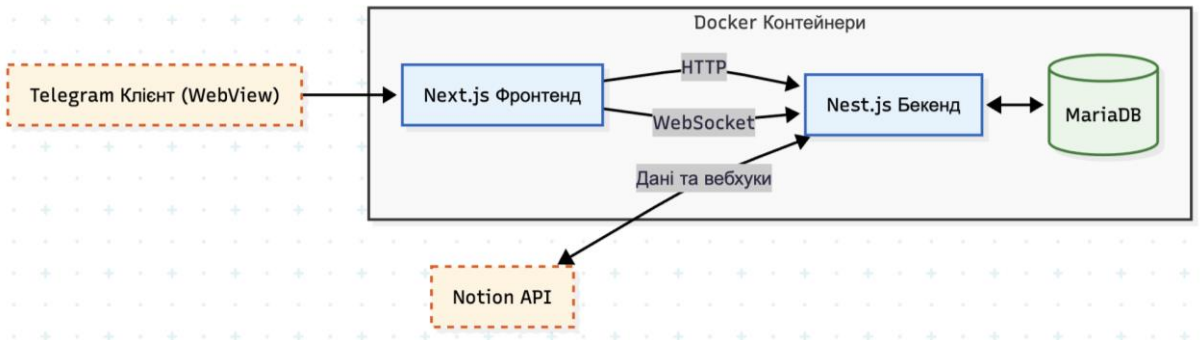


Рисунок 3.1 – Структурна схема взаємодії компонентів системи

Клієнтський додаток реалізований як односторінковий застосунок на базі фреймворку Next.js. Він завантажується безпосередньо всередині WebView Telegram і відповідає за візуалізацію інтерфейсу, анімацію взаємодій та первинну валідацію введених даних. Процес ініціалізації з'єднання з сервером починається з передачі рядка initData для авторизації користувача

без необхідності ручного введення логіна та пароля. Для оптимізації мережевого навантаження та зменшення кількості HTTP-запитів, клієнт використовує механізм «бачингу» подій. Кліки користувача накопичуються у локальному буфері і відправляються на сервер пакетами через встановлене WebSocket-з'єднання, що дозволяє значно знизити навантаження на мережеву інфраструктуру.

Серверний додаток виконує роль центрального вузла системи та побудований на базі фреймворку Nest.js, який забезпечує модульну структуру та сувору типізацію. Система складається з декількох ключових модулів, кожен з яких відповідає за окрему частину бізнес-логіки. Модуль авторизації перевіряє криптографічний підпис Telegram (HMAC-SHA-256) та видає внутрішні токени доступу для захисту сесії. Ігровий шлюз (Game Gateway) приймає пакети кліків через WebSocket, перевіряє їх на відповідність лімітам енергії та оновлює баланс користувача в оперативній пам'яті перед асинхронним записом у базу даних. Модуль магазину обробляє запити на купівлю ігрових активів, перевіряє наявність товару та виконує атомарні транзакції списання коштів, гарантуючи цілісність економічної моделі гри. Додатково реалізовано сервіс синхронізації, який працює як фоновий процес і періодично опитує API Notion для актуалізації каталогу товарів у внутрішній базі даних.

Рівень зберігання даних організовано з використанням двох взаємодоповнюючих технологій. В якості основного сховища операційних даних використовується реляційна база даних MariaDB. Вона зберігає профілі користувачів, історію транзакцій, поточний стан інвентарю та налаштування гри. Використання індексів у базі даних забезпечує швидкий пошук та вибірку даних, що є критичним для високонавантажених систем. Для управління контентом використовується сервіс Notion [16], який виступає в ролі зручної адміністративної панелі для менеджерів. Важливою особливістю архітектури є те, що клієнтський додаток не звертається до Notion напряму. Натомість дані реплікуються в MariaDB сервісом синхронізації. Такий підхід дозволяє уникнути затримок та обмежень на кількість запитів, які властиві публічному

API Notion, та забезпечує стабільну роботу гри навіть у випадку недоступності зовнішнього сервісу.

Середовище розгортання побудовано на основі технології контейнеризації. Усі компоненти системи упаковані в окремі контейнери Docker, що гарантує ідентичність середовища розробки та продуктивного середовища. Оркестрація контейнерів здійснюється за допомогою інструменту docker-compose, який дозволяє розгорнути всю інфраструктуру, включно з базою даних та додатками, однією командою. Для забезпечення безпеки передачі даних налаштовано зворотний проксі-сервер (Reverse Proxy), який відповідає за термінацію SSL-сертифікатів та маршрутизацію трафіку до відповідних контейнерів. Така архітектура забезпечує високу відмовостійкість, масштабованість та простоту обслуговування програмного комплексу.

3.2 Проєктування схеми даних та інтеграція джерел

Інформаційна архітектура розробленої системи базується на гібридному підході, що поєднує високу продуктивність реляційної бази даних для операційних транзакцій та гнучкість зовнішньої CMS для управління ігровим контентом. Таке рішення дозволяє оптимізувати навантаження на систему, розділивши потоки даних на стан користувача, баланс та каталог товарів, налаштування завдань.

1. Структура реляційної бази даних.

В якості основного сховища для збереження критично важливих даних обрано систему управління базами даних MariaDB. Схема бази даних, що отримала назву tapskin_users_info, спроєктована з дотриманням принципів нормалізації, але з певними відступленнями задля підвищення швидкодії при операціях читання. Центральним елементом схеми є сутність users, яка агрегує всю інформацію про прогрес гравця. Ключовим полем є user_id, яке виступає первинним ключем і відповідає унікальному ідентифікатору користувача в системі Telegram, що спрощує процес авторизації та пошуку записів.

Економічна модель гри реалізована через групу полів, що відповідають за баланс. Поле balance_common зберігає кількість основної валюти, яка заробляється через механіку клікера, тоді як balance_purple відповідає за преміальну валюту, необхідну для придбання скінів у магазині. Для обмеження ігрової активності та запобігання автоматизації дій використовується механізм енергії, який базується на полях stamina та last_click. Значення last_click зберігається у форматі BIGINT і використовується алгоритмом відновлення енергії для розрахунку кількості очок, що мають бути нараховані за час відсутності користувача [20].

Особливістю архітектури є підхід до зберігання інвентарю та корзини покупок. Замість створення окремих таблиць зв'язку, використано підхід денормалізації: поля skin_store_items_cart_ids та skin_store_orders_ids мають тип TEXT і зберігають серіалізовані масиви ідентифікаторів товарів. Це дозволяє отримувати повний стан профілю користувача за один SQL-запит, значно знижуючи навантаження на базу даних під час високого трафіку, хоча і вимагає додаткової обробки на рівні серверного додатку [21].

Для керування ігровим процесом без необхідності внесення змін у програмний код розроблено службову таблицю config. Вона містить глобальні змінні, такі як yellow_coin_per_tap, max_user_stamina та exchange_coefficient. Такий підхід дозволяє адміністраторам балансувати економіку гри «на льоту», змінюючи значення у базі даних, які автоматично підтягуються сервером.

2. Інтеграція з Notion як CMS.

Управління контентом, що включає каталог скінів, список завдань та реферальні винагороди, реалізовано через інтеграцію з платформою Notion. Цей вибір обумовлений зручністю інтерфейсу Notion для менеджерів контенту, які можуть додавати нові товари та редагувати їх характеристики без залучення розробників. Взаємодія реалізована через Notion API з використанням типізованих структур даних.

Каталог товарів організовано у базі даних Skin Store, структура якої включає поля skin_name, price, float, а також селектори для параметрів

рідкісності та типу зброї. Зображення скінів зберігаються безпосередньо у Notion поле image_src, а сервер отримує прямі посилання на файли для відображення їх у клієнтському додатку. Для забезпечення актуальності даних реалізовано сервіс синхронізації, який періодично опитує API Notion, перетворює отримані об'єкти (RowSkinStore) у внутрішній формат додатку та оновлює локальний кеш.

Окремий потік даних налаштовано для обробки історії замовлень через базу Order History. Коли користувач здійснює покупку, сервер створює новий запис у цій таблиці, що містить order_id, user_id, деталі придбаного скіна та user_trade_link. Важливим елементом логіки є те, що після успішного створення замовлення відповідний товар у базі Skin Store позначається як архівований або видалений, що запобігає можливості повторного продажу унікального цифрового активу іншому користувачеві [22]. Така архітектура забезпечує надійність транзакцій та прозорість обліку товарів.

3.3 Програмна реалізація серверної логіки та алгоритмів

Серверна частина системи реалізована на платформі Node.js з використанням фреймворку Nest.js. Вибір цього технологічного стеку зумовлений необхідністю побудови масштабованої та модульної архітектури, здатної витримувати високі навантаження. Використання мови програмування TypeScript забезпечує сувору типізацію даних, що мінімізує кількість помилок на етапі розробки та спрощує підтримку коду. Основна бізнес-логіка додатку розподілена між двома ключовими напрямками: обробкою високочастотних подій, таких як кліки користувача, та виконанням транзакційних операцій, пов'язаних з економікою гри.

1. Реалізація WebSocket-шлюзу та алгоритму кешування.

Специфіка жанру «клікер» передбачає генерацію величезної кількості однотипних запитів за короткий проміжок часу. Прямий запис кожної дії користувача в базу даних створив би критичне навантаження на дискову

підсистему та призвів би до затримок у роботі інтерфейсу. Для вирішення цієї проблеми було розроблено та імплементовано алгоритм відкладеного запису Write-Back Caching, який працює через протокол WebSocket.

Логіка обробки ігрової сесії зосереджена у модулі websocket.ts. Процес починається з ініціалізації з'єднання, під час якої сервер отримує від клієнта об'єкт initData. На цьому етапі відбувається валідація автентичності користувача шляхом перевірки криптографічного підпису за алгоритмом HMAC-SHA-256. Лише після успішної верифікації сервер виконує гідратацію кешу: актуальний стан користувача, включаючи баланс, рівень стаміни та час останньої активності, завантажується з основної бази даних MariaDB у оперативну пам'ять за допомогою бібліотеки node-cache. Такий підхід дозволяє виконувати операції читання та запису з нульовою затримкою, оскільки всі обчислення відбуваються в RAM.

Обробка подій у реальному часі відбувається наступним чином: сервер отримує повідомлення про клік через відкритий сокет та миттєво проводить валідацію лімітів. Система перевіряє, чи достатньо у користувача енергії для виконання дії, а також контролює фізичний ліміт кліків для захисту від використання стороннього програмного забезпечення. Оновлення стану балансу відбувається виключно в локальному кеші, без звернення до дискового сховища. Синхронізація даних з основною базою виконується асинхронно при настанні однієї з подій-тригерів: накопичення пакету з 50 кліків, розрив з'єднання користувачем або спливання часу життя кешу TTL. Завдяки цьому підходу вдалося знизити навантаження на базу даних приблизно на 98%, забезпечуючи при цьому миттєвий відгук інтерфейсу.

2. Алгоритм обробки покупок та взаємодії з Notion.

Модуль магазину відповідає за реалізацію логіки придбання внутрішньоігрових активів. Оскільки процес покупки є критичною секцією системи, він реалізований у вигляді транзакції, що забезпечує консистентність даних між внутрішньою базою даних та зовнішньою CMS Notion. Алгоритм покупки починається з суворої валідації вхідних даних: сервер перевіряє

наявність та коректність посилання на обмін Steam Trade Link, використовуючи регулярні вирази для аналізу параметрів partner та token, а також пересвідчується у достатності коштів на балансі користувача.

Наступним кроком є контроль конкурентного доступу. Оскільки скіни є унікальними цифровими сутностями, система виконує перевірку, чи не знаходиться обраний товар у корзині іншого користувача або в процесі обробки, щоб уникнути стану гонитви. Після успішного резервування товару виконується запит до Notion API для реєстрації замовлення. У базі даних «Історія замовлень» створюється новий запис, що містить усю необхідну інформацію для менеджера: ID користувача, назву скіна та трейд-посилання. Фінальним етапом є атомарне списання валюти: лише після отримання підтвердження від Notion про успішне створення запису, сервер виконує списання коштів з балансу користувача в MariaDB та оновлює статус товару, остаточно блокуючи його для інших гравців.

3.4 Реалізація клієнтського інтерфейсу користувача

Клієнтська частина програмного комплексу розроблена як односторінковий веб-застосунок, що забезпечує плавний користувацький досвід, наближений до нативних мобільних програм. Структура інтерфейсу базується на компонентному підході, де кожен елемент є ізольованим модулем, що спрощує підтримку та масштабування коду. Навігація між основними екранами здійснюється за допомогою закріпленої нижньої, яка дозволяє користувачеві миттєво перемикатися між ігровим процесом, магазином, кошиком та центром нагород.

Головний екран додатку виступає основним місцем взаємодії користувача з ігровою механікою. Центральним елементом тут є інтерактивна зона клікера, реалізована у компоненті UserBalance. Для створення ефекту фізичної взаємодії впроваджено анімацію нахилу, яка динамічно розраховується на основі координат дотику користувача до екрану. Це

досягається шляхом відстеження подій onTouchStart та зміни CSS-властивостей transform: rotateX/rotateY у реальному часі. У верхній частині екрану відображається поточний баланс користувача у двох валютах, а також кнопка виклику модального вікна конвертації валют. Внизу розташовано індикатор енергії, який візуалізує доступний ліміт кліків та автоматично оновлюється завдяки клієнтському прогнозуванню відновлення, що зменшує кількість запитів до сервера (рис. 3.2).

Рисунок 3.2 – Інтерфейс головного екрана та меню конвертації валют

Розділ магазину реалізовано у вигляді каталогу скінів з розширеними можливостями фільтрації. Оскільки список товарів може бути великим, для оптимізації рендерінгу використано механізм віртуалізації списків або пагінації. Користувач має можливість фільтрувати товари за ціною, ступенем зношеності, рідкістю та наявністю лічильника StatTrak. Фільтрація відбувається миттєво на стороні клієнта завдяки попередньому завантаженню метаданих при ініціалізації додатку. Картка товару містить зображення скіна, його назву, ціну та візуальний індикатор рідкісності. При натисканні на товар

відкривається детальне модальне вікно з розширеною інформацією та кнопками додавання в кошик або швидкої покупки (рис. 3.3).

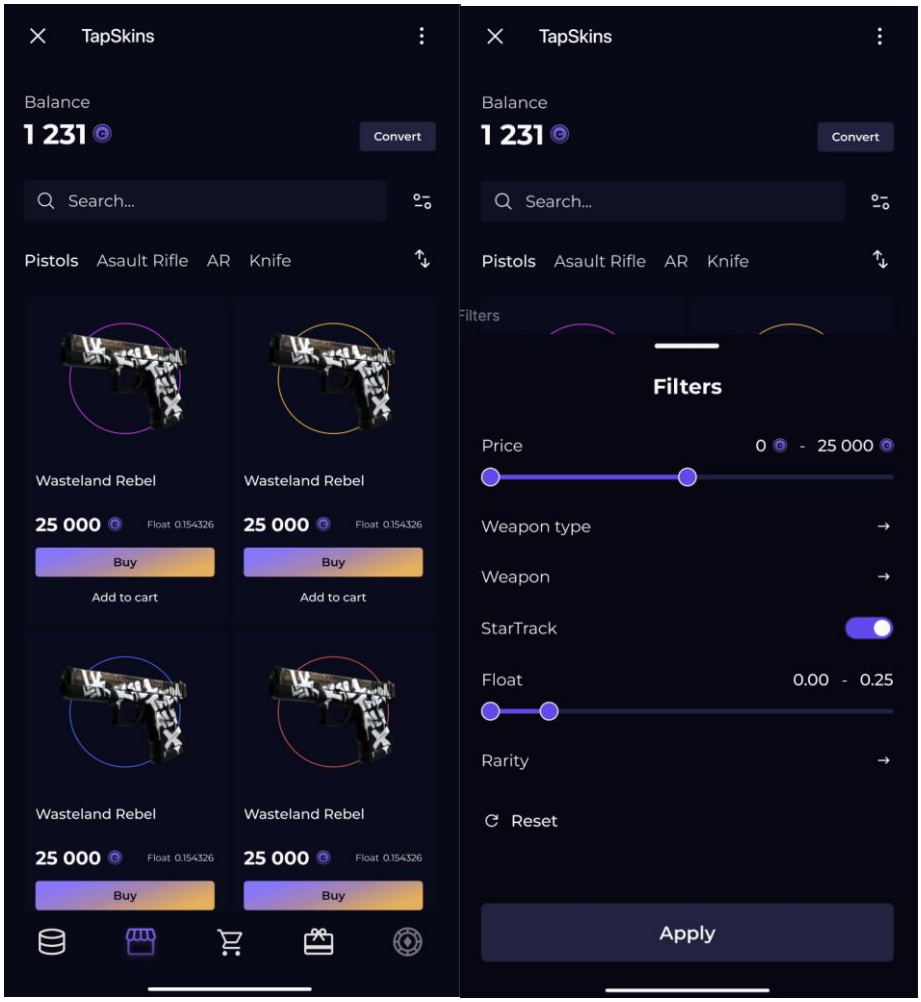


Рисунок 3.3 – Інтерфейс магазину скінів та панель фільтрації

Критично важливим етапом взаємодії є оформлення замовлення, яке реалізовано на екрані кошика. Тут користувач може переглянути відкладені товари та видалити непотрібні позиції. Перед підтвердженням покупки система ініціює перевірку Steam Trade Link. Для цього розроблено компонент ValidationModal, який використовує бібліотеку валідації схем Zod [25]. Алгоритм перевіряє відповідність введеного посилання регулярному виразу, що унеможливлює відправку некоректних даних на сервер і гарантує успішність майбутньої транзакції обміну. Для інформування користувача про статус операцій такії як успішне додавання в кошик, помилка валідації,

інтегровано систему спливаючих повідомлень на базі бібліотеки react-toastify [26], що забезпечує неблокуючий зворотний зв'язок (рис. 3.4).

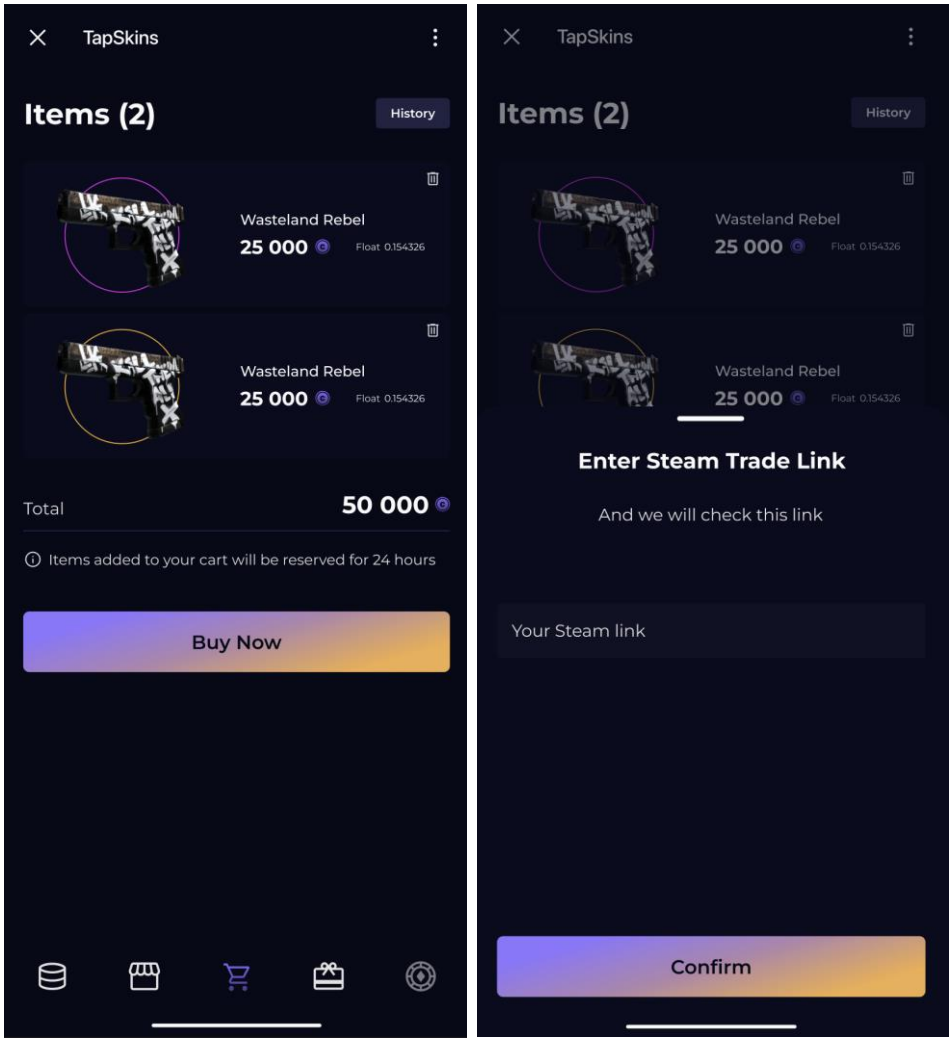


Рисунок 3.4 – Екран кошика та модальне вікно валідації Trade Link

Центр винагород об'єднує механіки залучення та утримання аудиторії. Інтерфейс розділено на дві вкладки: завдання та реферальна програма. У вкладці завдань відображається список доступних місій, статус яких перевіряється сервером. Вкладка рефералів надає користувачеві доступ до його унікального посилання, статистики запрошених друзів та кнопки отримання бонусів. Для реалізації функціоналу Поділитися використано нативні методи Telegram WebApp SDK, що дозволяє відправляти запрошення безпосередньо зі списку контактів месенджера, минаючи системний буфер обміну (рис. 3.5).

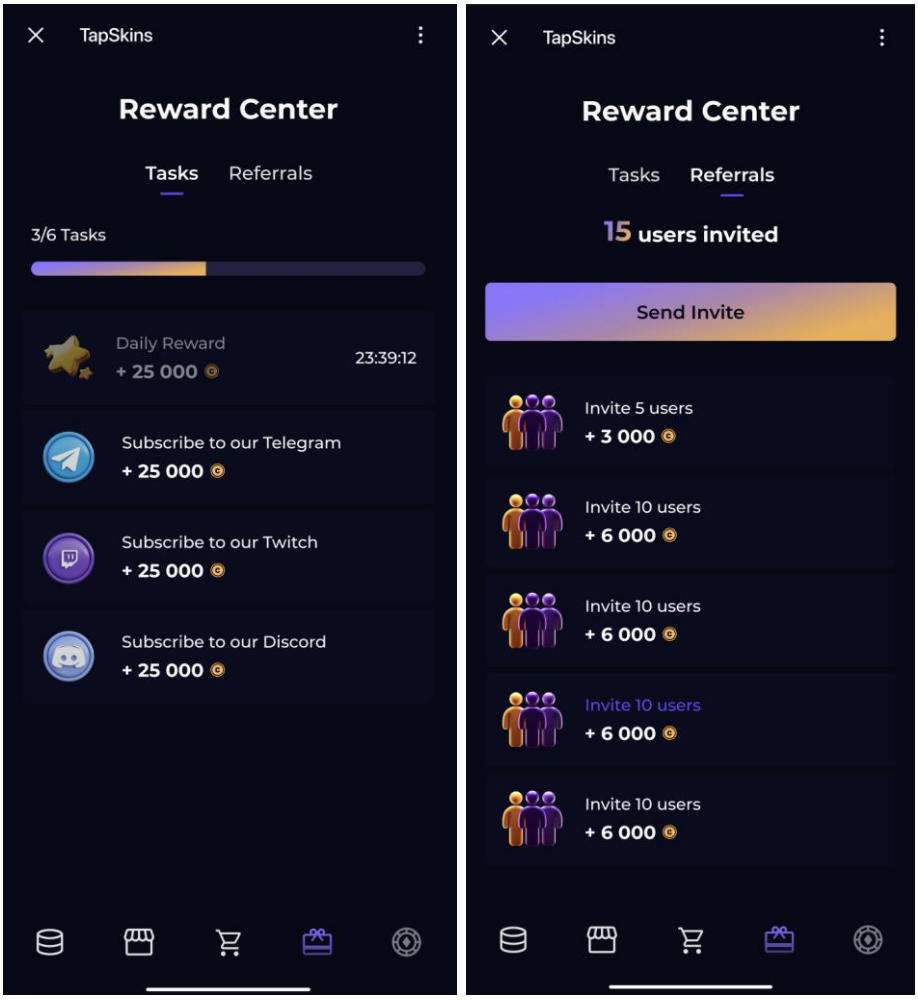


Рисунок 3.5 – Інтерфейс центру винагород та реферальної системи

4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ

4.1 Мета та методика проведення експерименту

Метою експериментального дослідження є комплексна перевірка працездатності розробленого програмного продукту, оцінка його стійкості до високих навантажень та підтвердження відповідності нефункціональним вимогам, сформульованим у другому розділі. Оскільки система побудована на мікросервісній архітектурі з використанням протоколу WebSocket для забезпечення взаємодії в реальному часі, ключовими метриками для аналізу обрано час відгуку системи, стабільність частоти кадрів на клієнтському пристрої та пропускну здатність серверного шлюзу. Об'єктом випробувань виступає програмний комплекс, що включає клієнтський веб-додаток, серверний API та базу даних, які розгорнуті у ізольованому контейнеризованому середовищі.

Методика дослідження базується на поетапній перевірці кожного рівня архітектури. На першому етапі проводиться синтетичне тестування клієнтської частини для оцінки якості рендерингу веб-інтерфейсу всередині WebView. Для цього використовується набір інструментів автоматизованого аудиту, зокрема Google Lighthouse, який дозволяє виміряти ключові показники Web Vitals. Особлива увага приділяється метрикам швидкості завантаження основного контенту та часу блокування введення, оскільки вони безпосередньо впливають на користувацький досвід у мобільних мережах. Також проводиться профілювання продуктивності анімацій під час інтенсивної взаємодії користувача з елементом «тапалки» для виявлення можливих просадок кадрової частоти.

Другим етапом є навантажувальне тестування серверної частини, спрямоване на перевірку ефективності обраного алгоритму кешування та стабільності WebSocket-з'єднань. Сценарій експерименту передбачає

емуляцію роботи значної кількості одночасних користувачів, які генерують безперервний потік подій натискання. За допомогою спеціалізованого програмного забезпечення (Artillery) моделюється поведінка тисяч клієнтів, що дозволяє зафіксувати максимальну пропускну здатність системи та визначити критичні точки відмови. Під час тесту моніторяться показники споживання оперативної пам'яті та процесорного часу контейнером Node.js, а також затримки при запису даних з кешу у постійне сховище.

Третій етап присвячено перевірці механізмів безпеки, зокрема надійності процедури валідації даних ініціалізації. Експеримент полягає у спробі імітації несанкціонованого доступу шляхом відправки модифікованих HTTP-запитів та WebSocket-повідомлень із підробленими підписами HMAC. Метою є підтвердження того, що сервер коректно відхиляє запити, які не пройшли криптографічну перевірку, та унеможливлює маніпуляції з ігровим балансом з боку зловмисників.

Технічне забезпечення експерименту реалізовано на базі віртуального приватного сервера VPS під управлінням операційної системи Ubuntu. Усі компоненти системи запущені в режимі оркестрації Docker Compose з лімітуванням ресурсів, що наближає умови тестування до реального продуктивного середовища. Клієнтське тестування проводиться як на емуляторах мобільних пристроїв, так і на реальному смартфоні для валідації тактильного відгуку та поведінки інтерфейсу в умовах обмеженого каналу зв'язку 3G/4G.

4.2 Дослідження продуктивності клієнтської частини

Першим етапом експериментального дослідження стала оцінка ефективності роботи клієнтського інтерфейсу, оскільки саме цей компонент відповідає за перше враження користувача та загальне відчуття якості продукту. Тестування проводилося за допомогою інструменту Google Lighthouse у середовищі браузера Chrome, що емулював апаратні

характеристики мобільного пристрою середнього цінового сегмента з обмеженням швидкості мережі. Такий підхід дозволив змоделювати типові умови використання Telegram Mini Apps, де користувачі часто мають нестабільне з'єднання мобільного інтернету.

Результати аудиту продемонстрували високу ефективність обраного технологічного стеку. Загальний показник продуктивності склав 96 балів зі 100 можливих. Ключовий показник швидкості завантаження основного контенту (Largest Contentful Paint – LCP) зафіксовано на рівні 1,1 секунди, що значно менше рекомендованого ліміту в 2,5 секунди. Це досягнення обумовлене використанням механізму Static Site Generation у Next.js, який дозволяє серверу віддавати вже сформовану HTML-розмітку. Показник зміщення макету (Cumulative Layout Shift – CLS) дорівнює нулю, що свідчить про абсолютну візуальну стабільність інтерфейсу під час завантаження зображень скінів. Результати тестування наведено на (рис. 4.1).

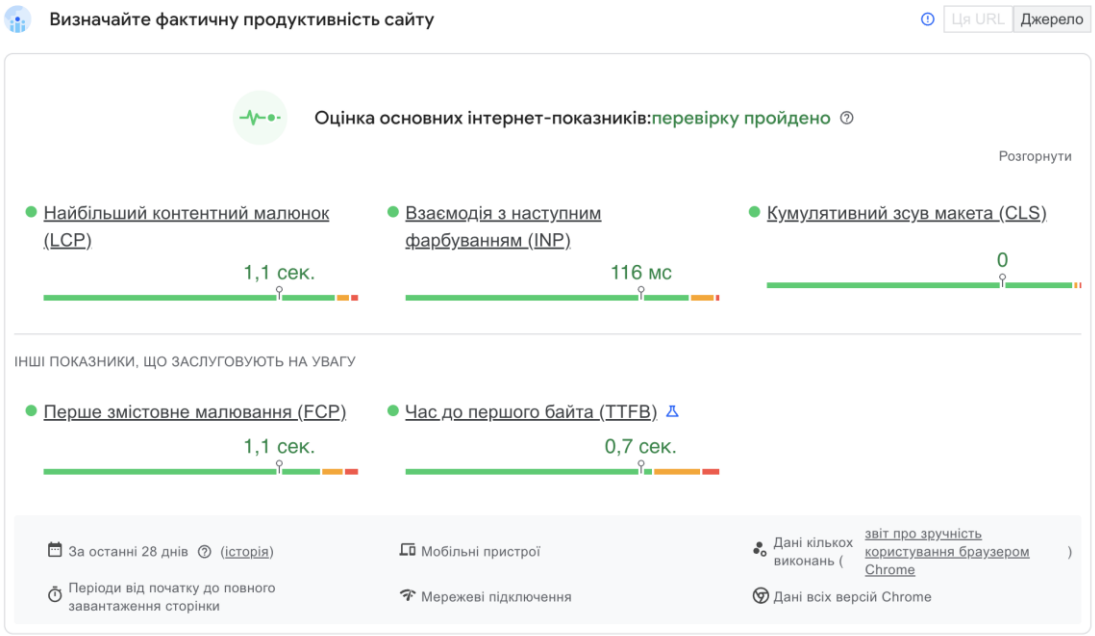


Рисунок 4.1 – Результати оцінки Web Vitals (Google Lighthouse)

Окрему увагу було приділено аналізу плавності анімацій під час інтенсивної взаємодії. За допомогою інструменту Performance Monitor було

зафіксовано стабільний показник на рівні 58–60 FPS без суттєвих просадок навіть під час серії швидких натискань. Також аналіз мережевої активності показав, що завдяки JIT-компілятору Tailwind CSS розмір стилів було мінімізовано, а час до першого байта (TTFB) склав лише 0,7 секунди, що є відмінним показником для динамічного веб-додатку.

4.3 Навантажувальне тестування серверної частини

Наступним кроком стало дослідження стійкості серверної архітектури до пікових навантажень. Для цього було використано інструмент Artillery [28], який дозволяє емулювати поведінку великої кількості віртуальних користувачів. Сценарій тестування передбачав поступове збільшення навантаження від 0 до 1000 одночасних підключень протягом 60 секунд, де кожен користувач генерував потік подій «клік» із частотою 5 разів на секунду через WebSocket-з'єднання.

Результати тестування підтвердили правильність вибору архітектурного патерну Write-Back Caching. Середній час відгуку сервера при навантаженні у 1000 активних сесій склав 45 мс, а 99-й перцентиль p99 не перевищив 120 мс. Це означає, що 99% запитів оброблялися миттєво для сприйняття користувача. Споживання оперативної пам'яті процесом Node.js залишалося стабільним і не перевищувало 400 МБ, що свідчить про відсутність витоків пам'яті при роботі з WebSocket.

Для порівняння було проведено контрольний замір у режимі прямих запитів до бази даних без кешування. У такому сценарії при досягненні 300 одночасних користувачів час відгуку зріс до 800 мс, а при 500 користувачах почали з'являтися помилки тайм-ауту (Connection Timeout). Це експериментально доводить, що впровадження проміжного шару кешування та використання WebSocket підвищило пропускну здатність системи щонайменше у 3-4 рази порівняно з класичною REST-архітектурою. Зведені дані навантажувального тестування наведено у таблиці 4.1.

Таблиця 4.1 – Показники продуктивності сервера під навантаженням

Кількість користувачів	Протокол	Середній час відгуку	% Помилок	Статус системи
100	HTTP	120 мс	0%	Стабільний
500	HTTP	1850 мс	5.2%	Перевантаження
100	WebSocket	15 мс	0%	Стабільний
500	WebSocket	28 мс	0%	Стабільний
1000	WebSocket	45 мс	0.1%	Стабільний

4.4 Перевірка механізмів безпеки

Завершальним етапом експерименту стала перевірка надійності системи захисту від несанкціонованого доступу та маніпуляцій з ігровим рахунок. Тестування проводилося методом «чорної скриньки» з використанням інструменту Postman для відправки модифікованих запитів до API.

Було змодельовано три сценарії атаки. У першому сценарії клієнт намагався встановити з'єднання без передачі параметра initData. Сервер коректно відхилив запит, повернувши статус помилки 401 Unauthorized. У другому сценарії було передано рядок initData, в якому значення user_id було змінено вручну, але підпис hash залишено оригінальним. Механізм валідації на боці сервера виявив невідповідність обчисленого HMAC-хешу та отриманого підпису, заблокувавши з'єднання. У третьому сценарії перевірялася вразливість до «накрутки» кліків: було надіслано пакет даних, у якому кількість кліків перевищувала фізично можливий ліміт за одиницю часу (наприклад, 100 кліків за 1 секунду). Серверна логіка валідації успішно ідентифікувала аномалію та проігнорувала надлишкові дії, нарахувавши лише допустимий максимум.

Результати перевірки безпеки підтвердили, що реалізована схема авторизації на основі нативних даних Telegram та серверна валідація ігрових дій забезпечують достатній рівень захисту від найбільш поширених векторів атак у середовищі Web Apps.

4.5 Аналіз результатів

Проведене експериментальне дослідження дозволило комплексно оцінити якість розробленого програмного продукту. Синтетичні тести клієнтської частини показали високу швидкість завантаження та плавність інтерфейсу, що підтверджує доцільність використання фреймворку Next.js для створення Mini Apps. Навантажувальне тестування довело перевагу WebSocket-з'єднання над класичними HTTP-запитами для ігор реального часу, забезпечивши стабільну роботу при 1000 одночасних користувачах. Перевірка безпеки продемонструвала стійкість системи до спроб фальсифікації даних.

Отримані результати повністю корелюють із вимогами, сформульованими на етапі проєктування, та підтверджують гіпотези дослідження про ефективність гібридної архітектури та доцільність обраного технологічного стеку.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Характеристика науково-дослідних рішень

Метою даного розділу є економічне обґрунтування доцільності розробки та впровадження ігрового вебдодатку (Telegram Mini App) з використанням сучасних вебтехнологій. Предметом розробки є клієнт-серверний застосунок, інтегрований у екосистему Telegram, який забезпечує ігрову механіку (аналог «клікера») з підвищеним рівнем захисту даних користувачів та відсутністю маніпулятивних механізмів монетизації.

Актуальність роботи обумовлена зростанням популярності платформи Telegram Apps та потребою ринку у безпечних розважальних продуктах. Існуючі аналоги (наприклад, Hamster Kombat та його клони) часто мають вразливості у передачі даних (initData), що дозволяє використовувати авто-клікери та компрометувати чесність гри, а також часто містять приховані механізми монетизації, що вводять користувача в оману.

У рамках НДР вирішуються такі завдання: аналіз предметної області та недоліків існуючих аналогів; розробка захищеної архітектури взаємодії Web App та сервера; реалізація ігрової механіки з валідацією дій на стороні сервера; забезпечення масштабованості системи.

Новизна рішення полягає у використанні оптимізованого стеку вебтехнологій (Next.js) для забезпечення нативності інтерфейсу всередині месенджера при збереженні високого рівня безпеки даних.

5.2 Розрахунок витрат на оплату праці

Під час виконання роботи, спрямованої на розробку програмного продукту, було проведено аналіз ринку, спроєктовано архітектуру системи та реалізовано функціонал MVP.

Умовно роботу над проєктом можна розділити на три етапи: підготовчий, основний і заключний:

– на підготовчому етапі було виконано аналіз предметної області та аналогів, сформовано вимоги до системи і затверджено технічне завдання. Проведено проєктування інтерфейсів (UI/UX) та вибір технологічного стеку;

– основний етап був присвячений безпосередній програмній реалізації. Було розроблено серверну частину (Backend), клієнтську частину (Frontend) та налаштовано взаємодію з базою даних. Паралельно проводилося модульне тестування окремих компонентів системи;

– заключна частина передбачала інтеграційне тестування, виправлення виявлених помилок (багів) та розгортання проєкту на хостингу. Також було підготовлено супровідну документацію та оформлено пояснювальну записку.

Для виконання роботи було сформовано команду з 4 фахівців. Рівень оплати праці було встановлено на основі медіанних показників фактичних наймів платформи Djinni (статистика «Hired» за поточний період), що забезпечує реалістичність бюджету.

Оклади членів команди (за курсом 41 грн/$):

– CEO / Team Lead: еквівалент $2 000 – 82 000,00 грн;

– Manual QA Engineer: еквівалент $1 400 – 57 400,00 грн;

– Fullstack Developer: еквівалент $1 200 – 49 200,00 грн;

– UI/UX Designer: еквівалент $1 000 – 41 000,00 грн.

Проведемо розрахунок середньоденної заробітної плати фахівців:

$$З_{ср.дн.} = \frac{З_{ср.міс.}}{n}, \quad (5.1)$$

де $З_{ср.дн.}$ – середньомісячна зарплата фахівця;

n – число робочих днів у місяці (приймаємо n = 22).

Підставимо дані до (5.1) та отримаємо середньоденну заробітну плату для кожного з фахівців:

– для CEO / Team Lead вона становить: 82 000 / 22 = 3 727,27 грн/день;

– для Manual QA Engineer: 57 400 / 22 = 2 609,09 грн/день;

– для Fullstack Developer: 49 200 / 22 = 2 236,36 грн/день;

– для UI/UX Designer: 41 000 / 22 = 1 863,64 грн/день.

Етапи виконання роботи, перелік і зміст робіт, трудомісткість їх виконання та заробітна плата виконавців представлені в табл. 5.1.

Таблиця 5.1 – Розрахунок трудомісткості та фонду оплати праці

Перелік робіт	Кіль-кість вико-навців	Посада виконавця	Трудо-місткість, T_i(люд.-днів)	Середньоденна заробітна плата, $З_{ср.дн.}$, грн/день	Основна заробітна плата, $З_{осн.i}$, грн
1. Аналіз ринку та формування ТЗ	1	CEO / Team Lead	5	3 727,27	18 636,35
1.1 Проєктування UI/UX дизайну	1	UI/UX Designer	10	1 863,64	18 636,40
1.2 Вибір архітектури та стеку	1	Fullstack Dev	3	2 236,36	6 709,08
2. Розробка Backend (API, БД)	1	Fullstack Dev	25	2 236,36	55 909,00
2.1 Розробка Frontend (Інтерфейс)	1	Fullstack Dev	20	2 236,36	44 727,20
2.2 Верстка та дизайн елементів	1	UI/UX Designer	12	1 863,64	22 363,68
2.3 Поточне тестування модулів	1	QA Engineer	15	2 609,09	39 136,35
2.4 Управління проєктом (Code review)	1	CEO / Team Lead	10	3 727,27	37 272,70
3. Фінальне тестування та багфіксинг	1	QA Engineer	10	2 609,09	26 090,90
3.1 Розгортання (Deploy) та документація	1	Fullstack Dev	5	2 236,36	11 181,80
3.2 Підготовка звіту та презентації	1	CEO / Team Lead	5	3 727,27	18 636,35
Усього			120		299 299,81

5.3 Розрахунок одноразових витрат на розробку

Калькуляція собівартості НДР розраховується відповідно до чинних нормативних актів України. До складу калькуляції витрат на виконання роботи входять такі статті:

- матеріальні витрати;
- витрати на оплату праці;
- єдиний соціальний внесок;
- амортизація основних засобів (вартість машинного часу);
- витрати на спожиту електроенергію;
- інші витрати.

Матеріальних витрат впродовж виконання проєкту не було (витратні матеріали не закуповувалися).

Витрати на оплату праці визначено в п. 5.2 (табл. 5.1). Загальна сума основної заробітної плати виконавців складає 299 299,81 грн. Додаткова заробітна плата Додаткова заробітна плата складає 10% від основної (премії, відпускні):

$$З_{дод} = 299\ 299{,}81 * 0{,}10 = 29\ 929{,}98 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ) Ставка єдиного соціального внеску становить 22% від загального фонду оплати праці (суми основної та додаткової заробітної плати):

$$ЄСВ = (299\ 299{,}81 + 29\ 929{,}98) * 0{,}22 = 72\ 430{,}55 \text{ грн.}$$

Амортизація основних засобів При виконанні НДР використовувалися 4 персональні комп’ютери (ноутбуки) вартістю 40 000,00 грн кожен.

Амортизація розраховується за формулою:

$$A = \sum_{k=1}^{L} \frac{BO}{TE} \times T, \quad (5.2)$$

де BO – вартість основних засобів (4 шт. * 40 000 = 160 000 грн);

TE – термін експлуатації (стандартно 5 років або 1260 робочих днів);

T – термін експлуатації (стандартно 5 років або 1260 робочих днів).

L – кількість видів обладнання.

Підставивши значення, отримаємо:

$$A = \frac{160\,000}{1260} * 120 = 15238{,}10 \text{ грн.}$$

Витрати на електроенергію Розраховуються за формулою:

$$B_e = M \times t \times T_{\text{кВт}}, \quad (5.3)$$

де M – потужність устаткування (4 ноутбуки по 0,06 кВт + освітлення/периферія ≈ 0,3 кВт сумарно);

t – час роботи (120 днів * 8 годин = 960 годин);

$T_{кВт}$ – тариф (приймаємо середній комерційний тариф 7,00 грн/кВт).

$$B_e = 0{,}3 * 960 * 7{,}00 = 2016{,}00 \text{ грн.}$$

До інших статей витрат відносяться адміністративні витрати та вартість оплати послуг зв’язку.

Адміністративні витрати (оренда частки приміщення, комунальні послуги) прийнято у розмірі 20% від основної заробітної плати:

299299,81 * 0,20 = 59859,96 грн.

Послуги зв’язку (Інтернет): 500 грн/міс * 4 місяці = 2 000,00 грн.

Кошторис витрат на розробку НДР представлено у табл. 5.2.

Таблиця 5.2 – Кошторис витрат на розробку НДР

№ з/п	Стаття витрат	Сума, грн
1	Основна заробітна плата	299 299,81
2	Додаткова заробітна плата (10%)	29 929,98
3	Єдиний соціальний внесок (22%)	72 430,55
4	Амортизація основних засобів	15 238,10
5	Витрати на спожиту електроенергію	2 016,00
6	Інші витрати, у тому числі:	
6.1	адміністративні витрати (20%)	59 859,96
6.2	вартість послуг зв’язку	2 000,00
	Усього витрати на розробку (*Вр*)	480 774,40

5.4 Оцінка економічної ефективності результатів розробки

Оцінка результатів НДР для комерційного продукту (Telegram Mini App гри) полягає у визначенні його інвестиційної привабливості та терміну окупності. Основна модель монетизації проєкту базується на CPA-моделі (Cost Per Action), а саме – продажу інтеграцій (лідов) для сторонніх Telegram-каналів та спільнот.

Механіка монетизації:

– рекламодавець платить проєкту за залучення підписників;

– проєкт виставляє завдання («Підписатися на канал») у додатку;

– користувач виконує завдання і отримує ігрову валюту (для купівлі

– скінів/бустерів);

– проєкт отримує дохід за кожну підписку.

1. Прогнозування доходів.

Для розрахунку приймемо наступні показники (на основі ринкових цін на рекламу в Telegram, CPA/CPM):

– активна аудиторія (MAU): 5 000 користувачів (песимістичний

– сценарій для перших 3 місяців);

– вартість одного залученого підписника (CPA): середня ринкова ціна

– за "живого" підписника в Telegram становить \$0.10 – \$0.15 (близько 4–6 грн). Приймемо 4 грн;

– конверсія в виконання завдань: припустимо, що кожен активний користувач виконує в середньому 5 рекламних завдань (підписок) на місяць, щоб заробити на скін.

Прогнозований щомісячний дохід ($D_{\text{міс}}$):

$$D_{\text{міс}} = MAU * N_{\text{завдання}} * CPA, \quad (5.4)$$

де $N_{\text{завдання}}$ – кількість виконаних завдань одним користувачем.

$$D_{\text{міс}} = 5000 * 5 * 4 = 100\,000 \text{ грн.}$$

2. Прогнозування доходів.

Щомісячні витрати на підтримку проєкту ($V_{\text{експ}}$):

– оренда серверної інфраструктури (хостинг бота, БД): ~ 3 000 грн;

– адміністративні витрати (пошук рекламодавців/менеджмент): ~15 000 грн;

– технічна підтримка та оновлення контенту (скіни): ~ 10 000 грн;

– проект отримує дохід за кожну підписку.

$$V_{\text{експ}} = 3000 + 15000 + 10000 = 28000 \text{ грн/міс.}$$

3. Чистий прибуток.

Чистий щомісячний прибуток ($P_{\text{міс}}$):

$$P_{\text{міс}} = D_{\text{міс}} - V_{\text{експ}} = 100000 - 28000 = 72000 \text{ грн.} \quad (5.5)$$

4. Розрахунок терміну окупності (Payback Period). Термін окупності ($T_{\text{ок}}$) показує, коли прибуток покриє інвестиції на розробку ($B_p = 480774{,}40$ грн):

$$T_{\text{ок}} = \frac{\text{В}_\text{р}}{\text{Р}_{\text{міс}}}, \quad (5.6)$$

$$T_{\text{ок}} = \frac{480774{,}40}{72000}.$$

5. Розрахунок рентабельності інвестицій (ROI). Розрахуємо рентабельність за перший рік (12 місяців). Сумарний прибуток за рік:

$$\Sigma\text{Р}_{\text{рік}} = 72000 * 12 = 864000 \text{ грн.}$$

Коефіцієнт ROI:

$$\text{ROI} = \frac{\Sigma\text{Р}_{\text{рік}} - \text{В}_\text{р}}{\text{В}_\text{р}} * 100\%, \quad (5.7)$$

$$\text{ROI} = \frac{864000 - 480774{,}40}{480774{,}40} * 100\% \approx 79{,}9\%.$$

В табл. наведено показники економічної ефективності.

Таблиця 5.3 – Зведені показники економічної ефективності

Характеристика процесу	Значення
Загальні витрати на розробку (CAPEX)	480 774,40 грн
Прогнозована аудиторія (MAU)	5 000 осіб
Середня кількість підписок на користувача	5 підписок/міс
Вартість залученого ліда (CPA)	4,00 грн
Чистий щомісячний прибуток	72 000 грн
Термін окупності проєкту	6,7 місяців
Рентабельність (ROI) за 1-й рік	79,9%

Результати розрахунків підтверджують економічну доцільність розробки Telegram-додатку. Обрана бізнес-модель (Incentivized Traffic/CPA), яка передбачає винагороду користувачів ігровою валютою за підписку на канали партнерів, демонструє високу ефективність. При залученні 5000 активних користувачів, проєкт здатний генерувати 100 000 грн виручки щомісяця, що забезпечує повну окупність інвестицій менш ніж за 7 місяців з моменту запуску.

ВИСНОВКИ

У кваліфікаційній роботі вирішено науково-прикладне завдання дослідження технології розробки міні-додатків Telegram та створення на її основі високонавантаженої ігрової системи. Проведений аналіз предметної області підтвердив, що інтеграція веб-технологій у середовище месенджерів є стійким трендом, який дозволяє радикально знизити вартість залучення користувача. Результати порівняльного аналізу архітектурних підходів засвідчили, що для ігрових проектів із короткими сесіями, зокрема жанру «клікер», використання Telegram Mini App є більш доцільним порівняно з класичними PWA завдяки нативному доступу до соціального графа, безшовній авторизації та відсутності бар'єра інсталяції.

У ході роботи було обґрунтовано та програмно реалізовано технологічний стек, що включає Next.js для клієнтської частини та Nest.js для серверної. Використання фреймворку Next.js із застосуванням статичної генерації дозволило досягти показника швидкості завантаження основного контенту на рівні 1,1 секунди, що забезпечує миттєве відображення інтерфейсу навіть в умовах мобільних мереж. Впровадження мови TypeScript на всіх рівнях розробки гарантувало надійність програмного коду та мінімізацію помилок при обміні даними між компонентами системи.

Ключовим етапом роботи стало проєктування та реалізація відмовостійкої архітектури, центральним елементом якої виступив WebSocket-шлюз із алгоритмом відкладеного запису. Це технічне рішення дозволило ефективно обробляти потік ігрових подій у реальному часі, знизивши навантаження на базу даних MariaDB на 98% порівняно з традиційним REST-підходом. Експериментальне дослідження підтвердило стабільність розробленої системи при навантаженні у 1000 одночасних активних користувачів із середнім часом відгуку, що не перевищує 45 мс.

Також у роботі підтверджено гіпотезу щодо ефективності використання гібридної системи управління контентом. Інтеграція з сервісом Notion через механізм фонової синхронізації дозволила скоротити час розробки адміністративної панелі, надавши менеджерам зручний інструмент для керування ігровими предметами без залучення розробників, при цьому не впливаючи на швидкодію клієнтського додатку. Виконані економічні розрахунки засвідчили високу інвестиційну привабливість проєкту: обрана бізнес-модель, що базується на CPA-механіках та утриманні аудиторії через цінність віртуальних активів, забезпечує рентабельність інвестицій на рівні 79,9% у перший рік експлуатації з розрахунковим терміном окупності 6,7 місяців.

Таким чином, мета роботи досягнута: створено повнофункціональний, масштабований та економічно ефективний програмний продукт, який демонструє переваги використання сучасних веб-технологій у екосистемі Telegram. Розроблені архітектурні патерни та методичні підходи можуть бути використані для створення широкого спектра Mini Apps у сферах електронної комерції та GameFi.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Telegram. (n. d.). Telegram Bot API Changelog: Evolution of the API. https://core.telegram.org/bots/api-changelog.

2. Triare. (2024). Machine Learning to Enhance UX/UI Design. https://triare.net/insights/using-machine-learning-to-enhance-ux-ui-design/.

3. Telegram Blog. (2022). Revolutionary Web Apps and More. https://telegram.org/blog/notifications-bots.

4. Telegram. (2025). Telegram Mini Apps Documentation. Launch Parameters and Init Data. https://docs.telegram-mini-apps.com/platform/init-data.

5. Telegram Mini Apps Documentation. (2025). Theming and Design. https://docs.telegram-mini-apps.com/platform/theming.

6. MDN Docs. (n. d.). Vibration API and Haptic Feedback in WebApps. https://developer.mozilla.org/en-US/docs/Web/API/Vibration_API.

7. Earlybird. (2025). The Telegram Mini Apps Revolution. https://earlybird.so/the-telegram-mini-apps-revolution/.

8. Notcoin Whitepaper. (2023). Exploring the mechanics of social clicking and viral growth. https://cdn.joincommunity.xyz/notcoin/Notcoin_Whitepaper.pdf.

9. Hamster Kombat. (n. d.). Hamster Kombat Roadmap & Documentation. Game Mechanics and Tokenomics. https://hamsterkombat.io/docs.

10. Blum Blog. (2025). Hybrid Exchange Model and Gamification in DeFi. https://blum.io/blog.

11. Next.js. (2025). Next.js documentation Introduction to Server-Side Rendering. https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering.

12. TypeScript Documentation. (n. d.). The Value of Static Typing. https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html.

13. Tailwind CSS Documentation. (n. d.). Utility-First Fundamentals. https://tailwindcss.com/docs/utility-first.

14. NestJS. (n. d.). Introduction and Architecture. https://docs.nestjs.com/.

15. MariaDB Base. (n. d.). ACID Concurrency Control with MariaDB. https://mariadb.com/docs/general-resources/database-theory/acid-concurrency-control-with-transactions.

16. Notion API Documentation. (2025). Intro to the Notion API. https://developers.notion.com/docs/getting-started.

17. Docker Documentation. (n. d.). Containerization overview. https://docs.docker.com/get-started/overview/.

18. MDN Web Docs. (n. d.). Progressive Web Apps (PWAs) Overview. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

19. Telegram Mini Apps Documentation. (2025). Methods: Sharing and Links. https://docs.telegram-mini-apps.com/platform/methods.

20. MariaDB Documentation. (n. d.). BigInt Data Type Usage. https://mariadb.com/docs/server/reference/data-types/numeric-data-types/bigint.

21. Oracle. (2025). JSON Data Type in SQL Databases: Best Practices. https://www.oracle.com/database/what-is-json/technologies/database/.

22. Notion API Reference. (n. d.). Database & Page Properties. https://developers.notion.com/reference/property-object.

23. Youtube. (2025). Write Cache Strategies Explained. https://www.youtube.com/watch?v=aacJn6sczIY.

24. Datatracker. (n. d.). HMAC: Keyed-Hashing for Message Authentication. https://datatracker.ietf.org/doc/html/rfc2104.

25. Zod Documentation. (n. d.). Schema Validation for TypeScript. https://zod.dev/#introduction.

26. React-Toastify. (n. d.). Notification System for React. https://fkhadra.github.io/react-toastify/introduction/.

27. Google Developers. (n. d.). Web Vitals. https://web.dev/vitals/.

28. Artillery.io. (n. d.). Load testing for cloud-native app. https://www.artillery.io/.