МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕКОНОМІКИ І МЕНЕДЖМЕНТУ

Кафедра кібернетики та інформатики

КВАЛІФІКАЦІЙНА РОБОТА

освітній ступінь - «Бакалавр»

на тему: **РОЗРОБКА ГЕЙМІФІКОВАНОЇ ПЛАТФОРМИ НА БАЗІ TELEGRAM З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ БЛОКЧЕЙН**

Виконав: студент спеціальності
126 «Інформаційні системи та технології»

Черкашин Іван Олександрович

Керівник: ***В'юненко Олександр Борисович***
к.е.н., доцент кафедри кібернетики та інформатики

Рецензент: ***Ободяк Віктор Корнелійович***
к.т.н., доцент кафедри кібербезпки СумДУ

Суми – 2025

СУМСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет	Економіки і менеджменту
Кафедра	Кібернетики та інформатики
Освітній ступінь	*«Бакалавр»*
Спеціальність	*126 «Інформаційні системи та технології»* *ОП «Інформаційні системи та технології»*

Затверджую:
Завідувач кафедри Світлана АГАДЖАНОВА

«_____ « _____ 202____ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Черкашину Івану Олександровичу

1. Тема роботи: Розробка гейміфікованої платформи на базі Telegram з використанням технології блокчейн

2. База практичного дослідження: Сумський національний аграрний університет м. Суми

керівник роботи В'юненко Олександр Борисович к.е.н., доцент

затверджена наказом по університету від « » №

3. Строк подання студентом закінченого проекту (роботи) «26» червня 2025 року

4. Вихідні дані до проекту (роботи):

Вихідними даними для проекту слугували результати аналізу ринку GameFi та існуючих аналогів, офіційна технічна документація платформ Telegram і TON, а також ключові принципи програмної інженерії та геймдизайну.

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

- *виконати постановку задачі;*
- *вибрати середовище для реалізації;*
- *розробити технічне завдання;*
- *розробити програмний продукт відповідно до поставлених задач;*
- *розробити інструкцію для користувача.*

6. Дата видачі завдання: **« » 202_ р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проекту	Примітка
1	Визначення об'єкту, предмету дослідження, формулювання мети та задач кваліфікаційної роботи, складання плану	*січень 2024 р.*	
2	Підбір та вивчення літературних джерел, технічної документації	*лютий 2024 р.*	
3	Розробка технічного завдання	*березень 2024 р..*	
4	Розробка платформи	*лютий-квітень 2024 р.*	
5	Створення інструкції користування вебзастосунком для користувачів	*до 14 травня 2024 р.*	
6	Оформлення пояснювальної записки	*до 30 травня 2024 р..*	
7	Оформлення презентації	*до 1 червня 2024 р.*	
8	Опрацювання зауважень керівника, перевірка на автентичність	*5 червня 2024 р.*	
9	Підготовка доповіді та її попередній захист	*9 червня 2024 р.*	
10	Перевірка з академічної доброчесності кваліфікаційної роботи	*9 червня 2024 р.*	
11	Представлення кваліфікаційної роботи на кафедру, деканат	*13 червня 2024 р.*	
12	Захист кваліфікаційної роботи	*26 червня 2024р.*	

Студент _____ ***Іван ЧЕРКАШИН***
(підпис)

Керівник роботи _____ ***Олександр В'ЮНЕНКО***
(підпис)

Перевірку на автентичність проведено _____ ***Надія БАРАННІК***
(підпис)

Кваліфікаційна робота допущена до захисту _____ ***Маргарита ЛИШЕНКО***
(підпис)

АНОТАЦІЯ

Черкашин І.О. Розробка гейміфікованої платформи на базі Telegram з використанням технології блокчейн

Кваліфікаційна робота зі спеціальності 126 «Інформаційні системи та технології» ОП «Інформаційні системи та технології» Сумський Національний Аграрний Університет. Суми, 2025 р. – Рукопис.

Кваліфікаційна робота присвячена дослідженню та розробці архітектури для комплексної ігрової платформи, що функціонує в середовищі месенджера Telegram. У роботі розглядаються ключові аспекти конвергенції соціальних платформ та технології блокчейн, аналізуються сучасні економічні моделі в GameFi, такі як Play-to-Earn (P2E), та проектується архітектура для реалізації Action RPG у форматі Telegram Mini App.

Об'єктом дослідження є процес розробки та інтеграції клієнт-серверних ігрових веб-додатків у середовище соціальних платформ. Предметом дослідження є архітектурні рішення, моделі даних та програмні засоби для створення Action RPG на базі Telegram Mini Apps з інтегрованою P2E-економікою. Метою роботи є підвищення ефективності взаємодії користувачів з ігровими Web3-платформами шляхом усунення бар'єрів входу.

Проведено комплексне тестування ключових модулів розробленого прототипу, включаючи продуктивність графічного рушія в середовищі TMA та коректність виконання блокчейн-транзакцій у мережі The Open Network (TON). Результати тестування підтвердили життєздатність архітектури та відповідність системи функціональним вимогам.

Результатом роботи є програмний прототип гейміфікованої платформи, що поєднує глибокий ігровий процес жанру Action RPG з доступністю та віральністю месенджера Telegram. Розроблена система демонструє ефективний підхід до створення складних ігрових продуктів у суперап-екосистемах та надає користувачам зручний вхід у світ Web3-ігор.

SUMMARY

Cherkashyn I. O. Development of a gamified platform based on Telegram using blockchain technology.

Qualification Thesis for the specialty 126 "Information Systems and Technologies," EP "Information Systems and Technologies," Sumy National Agrarian University. Sumy, 2025 – Manuscript.

The qualification thesis is dedicated to the research and development of an architecture for a complex gaming platform that functions within the Telegram messenger environment. The work examines the key aspects of the convergence of social platforms and blockchain technology, analyzes modern economic models in GameFi, such as Play-to-Earn (P2E), and designs an architecture for the implementation of an Action RPG in the Telegram Mini App format.

The object of the study is the process of developing and integrating client-server gaming web applications into social platform environments. The subject of the study is the architectural solutions, data models, and software tools for creating an Action RPG based on Telegram Mini Apps with an integrated P2E economy. The goal of the work is to increase the efficiency of user interaction with Web3 gaming platforms by eliminating entry barriers.

Comprehensive testing of the key modules of the developed prototype was conducted, including the performance of the graphics engine in the TMA environment and the correctness of executing blockchain transactions on The Open Network (TON). The test results confirmed the viability of the architecture and the system's compliance with functional requirements.

The result of the work is a software prototype of a gamified platform that combines the deep gameplay of the Action RPG genre with the accessibility and virality of the Telegram messenger. The developed system demonstrates an effective approach to creating complex gaming products within super-app ecosystems and provides users with a seamless entry into the world of Web3 games.

ЗМІСТ

ВСТУП

Актуальність теми. Актуальність роботи зумовлена сучасними тенденціями у розвитку цифрових технологій, зокрема конвергенцією соціальних платформ та технології блокчейн в індустрії інтерактивних розваг. Це поєднання формує нові можливості для розробки та розповсюдження цифрових продуктів і визначає високу актуальність даного дослідження.

Перший аспект актуальності стосується розвитку месенджера Telegram як багатофункціональної платформи. З аудиторією понад 900 мільйонів користувачів, Telegram еволюціонував від засобу комунікації до дистрибуційної екосистеми завдяки технології Telegram Mini Apps (TMA). Ці вебдодатки, що працюють в інтерфейсі месенджера, надають розробникам ефективний інструмент дистрибуції, оскільки усувають для користувача необхідність встановлення програмного забезпечення з App Store чи Google Play. Успішні кейси, як-от проєкт Notcoin, продемонстрували ефективність такої моделі поширення.

Другий аспект полягає в активному зростанні ринку GameFi та економічної моделі Play-to-Earn (P2E). Ця модель надає гравцям право власності на ігрові активи шляхом їх токенізації (NFT), що дозволяє монетизувати ігровий час. Поєднання комплексних ігрових механік жанру Action RPG (розвиток персонажа, динамічні бої) з прозорістю та безпекою блокчейну створює технічні й економічні передумови для розробки нових ігрових продуктів.

Незважаючи на наявний потенціал, більшість існуючих TMA-ігор представлені казуальними жанрами. На ринку практично відсутні комплексні Action RPG зі складною 2D/3D графікою, що пов'язано з технічними обмеженнями середовища виконання TMA та відсутністю стандартизованих архітектурних рішень. Відповідно, існує науково-технічна задача з розробки архітектури для графічно вимогливих ігор у середовищі супер-додатків.

Мета і завдання дослідження. **Метою** роботи є підвищення ефективності взаємодії користувачів з ігровими платформами шляхом розробки та програмної

реалізації гейміфікованої платформи у форматі Action RPG на базі Telegram Mini Apps, що інтегрує технологію блокчейн для реалізації механіки Play-to-Earn (P2E).

Для досягнення поставленої мети в роботі сформульовано наступні завдання:

- Провести аналіз предметної області: ринку GameFi, архітектурних особливостей платформи Telegram Mini Apps, а також існуючих програмних аналогів.
- Розробити архітектуру інформаційної системи, що включає проєктування клієнт-серверної взаємодії, моделі бази даних та схеми інтеграції з блокчейн-мережею.
- Здійснити програмну реалізацію ключових модулів додатку: клієнтської частини з ігровою логікою та графікою, а також серверної частини для обробки бізнес-логіки.
- Провести тестування функціональності та продуктивності розробленої платформи.

Об'єктом дослідження є процес розробки та інтеграції клієнт-серверних ігрових вебдодатків у середовище соціальних платформ та месенджерів.

Предметом дослідження є моделі даних, архітектурні рішення, методи взаємодії компонентів та програмні засоби для створення Action RPG платформи на базі Telegram Mini Apps з інтегрованою P2E-економікою.

Методи дослідження. Для вирішення поставлених завдань використано методи системного аналізу для декомпозиції проблеми та визначення вимог, порівняльного аналізу для дослідження існуючих рішень та UML-моделювання на етапі проєктування. Основною парадигмою програмування є об'єктно-орієнтована. Працездатність системи перевірялася за допомогою модульного та інтеграційного тестування.

Технологічний стек проєкту обрано з урахуванням вимог до продуктивності, масштабованості та швидкості розробки:

- Клієнтська частина (Frontend): JavaScript, Three.js (3D-графіка), Pixi.js (2D-графіка та інтерфейси).
- Серверна частина (Backend): Python, FastAPI (API), Django (адміністративна панель), SQLAlchemy (ORM).
- База даних: PostgreSQL.
- Інтеграція з блокчейном: Взаємодія з API блокчейн-мереж (TON або EVM-сумісних) для управління NFT та токенами.

Апробація результатів кваліфікаційної роботи. Результати кваліфікаційної роботи апробовані на конференціях: 1) Всеукраїнської науково-практичної конференції здобувачів вищої освіти (25 квітня 2025 р.), Суми; 2) міжнародна конференція XI International Scientific and Practical Conference “SCIENCE AND TECHNOLOGY: CHALLENGES, PROSPECTS AND INNOVATIONS” (19-21 червня 2025 р.), Осака, Японія.

Публікації. За матеріалами кваліфікаційної роботи опубліковано 2 тези наукових конференцій.

Структура і обсяг роботи. Кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел з 20 найменувань, додатків. Основний текст викладений на 48 сторінках комп’ютерного тексту, робота містить 1 таблицю, 16 рисунків, 3 додатки.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

В ході роботи над кваліфікаційним проєктом проведено комплексний аналіз сучасного стану розвитку цифрових технологій, який показав процес глибокої технологічної конвергенції на перетині соціальних платформ, індустрії інтерактивних розваг та децентралізованих фінансів. Розробка гейміфікованої платформи на базі Telegram з інтеграцією блокчейн-технологій знаходиться в епіцентрі цих змін. Для обґрунтування актуальності та формулювання конкретних завдань роботи, проведено системний аналіз наукових публікацій, технічної документації та ринкових звітів за трьома ключовими напрямками: екосистеми міні-додатків (Mini Apps) як новий канал дистрибуції; еволюція ігрових економічних моделей Play-to-Earn (P2E); та порівняльний аналіз блокчейн-інфраструктур.

Першим напрямком аналізу стала екосистема Telegram Mini Apps (TMA). Встановлено, що традиційна модель розповсюдження мобільних додатків через централізовані магазини Apple App Store та Google Play стикається з низкою системних викликів. До них належать високі комісії (до 30%), жорсткі політики модерації, перенасиченість ринку, що робить просування (App Store Optimization) вкрай складним, та надвисока вартість залучення користувачів (Customer Acquisition Cost). У відповідь на ці бар'єри розвинулась концепція «супер-додатків», як-от WeChat, що створюють єдину екосистему сервісів, утримуючи користувача та підвищуючи його життєву цінність (Lifetime Value). Згідно з офіційною документацією [1], Telegram активно трансформується за цією моделлю. Ключовим інструментом є технологія TMA — повноцінних веб-додатків, що виконуються в ізольованому середовищі WebView всередині клієнта Telegram [2]. Це дозволяє розробникам використовувати стандартні веб-технології (HTML,

CSS, JavaScript), а спеціальний об'єкт Telegram.WebApp забезпечує міст для взаємодії з клієнтом месенджера: отримання даних користувача, налаштувань теми, керування елементами інтерфейсу.

Аналіз технічної документації API [1, 2] дозволив виділити фундаментальні переваги TMA:

- Безшовний онбординг: Усувається головний бар'єр — пошук та встановлення додатку. Користувач запускає платформу миттєво за посиланням у чаті, каналі або від бота.
- Уніфікована ідентифікація: Авторизація відбувається автоматично з використанням існуючого профілю Telegram, що позбавляє користувача необхідності створювати новий акаунт.
- Віральна дистрибуція: Глибока інтеграція з соціальними функціями месенджера (реферальні посилання, таблиці лідерів у групових чатах, можливість ділитися ігровими досягненнями) створює потужні механізми для органічного поширення, що значно знижує маркетингові витрати.

Ринкове підтвердження ефективності TMA отримано на прикладі проєктів Notcoin та Hamster Kombat, які, попри механічну простоту, залучили десятки мільйонів користувачів. Водночас аналіз ринку показав, що більшість існуючих ігор на платформі є казуальними. Це пов'язано з об'єктивними технічними викликами: середовище WebView накладає суттєві обмеження на продуктивність, особливо в частині рендерингу 3D-графіки в реальному часі та виконання складної ігрової логіки на стороні клієнта. Це створює значну ринкову нішу та технологічний виклик для розробки складних інтерактивних систем, таких як Action RPG.

Наступним ключовим елементом дослідження стала еволюція ігрових економік. Проаналізовано перехід від моделі "Pay-to-Play" до "Free-to-Play" (F2P), де внутрішньоігрові активи залишалися власністю розробника без реальної цінності поза грою. Встановлено, що технологія блокчейн пропонує новий підхід, заснований на концепції цифрової власності, де ігрові активи представлені у вигляді невзаємозамінних токенів (NFT) стандарту ERC-721 або ERC-1155. Ця ідея

лежить в основі моделі Play-to-Earn (P2E). Дослідження першої хвилі P2E-проектів (на прикладі Axie Infinity) виявило їхні системні недоліки. Зокрема, моделі з двома токенами (наприклад, AXS як токен управління та SLP як утилітарний) страждали від гіперінфляції утилітарного токена. Його емісія («крани») значно перевищувала механізми спалювання («стоки»), що призводило до постійного тиску на ціну та знецінення заробітку гравців. Це перетворювало ігровий процес на монотонну роботу ("grind"), що призводило до відтоку аудиторії при падінні дохідності.

Сучасні практики, описані в рамках гнучких методологій [19], свідчать про еволюцію моделі в бік "Play-and-Earn" (P&E), або GameFi 2.0. Ключова ідея полягає в тому, що на першому місці має бути захоплюючий ігровий процес, а блокчейн та P&E виступають як інструмент розширення ігрового досвіду. Створення стійкої економіки вимагає ретельного проєктування "стоків" — механізмів витрачання токенів (крафтинг та покращення предметів, ремонт екіпіровки, участь у турнірах, купівля косметичних предметів). Встановлено, що жанр Action RPG, з його основними циклами (вбивство монстрів → отримання здобичі → покращення екіпіровки), надає ідеальне підґрунтя для імплементації таких збалансованих P&E моделей.

Реалізація таких економік вимагає надійної інфраструктури. Тому окрему увагу приділено порівняльному аналізу блокчейн-платформ. Вибір визначається балансом між децентралізацією, безпекою та масштабованістю (т.зв. "блокчейн-трилема"), де для ігор критичними є масштабованість (висока кількість транзакцій за секунду, TPS) та низькі комісії. В рамках роботи проаналізовано дві категорії платформ:

The Open Network (TON): Згідно з її технічною документацією [3, 4, 5], ця платформа представляє особливий інтерес через глибоку інтеграцію з Telegram. Її архітектура, що базується на принципах шардингу та асинхронної взаємодії смарт-контрактів (модель акторів), теоретично дозволяє досягати мільйонів транзакцій за секунду. Наявність інтегрованого у Telegram гаманця TON Wallet та бібліотек, як-от TonWeb [6], значно спрощує інтеграцію та покращує досвід користувача.

EVM-сумісні блокчейни (BNB Chain, Polygon): Ці платформи [16, 17] є лідерами ринку GameFi завдяки величезній та зрілій екосистемі. Мова Solidity [14] є стандартом де-факто для смарт-контрактів, що забезпечує доступ до великої спільноти розробників. Існує безліч готових фреймворків (Hardhat, Truffle), інфраструктурних сервісів (IPFS для зберігання метаданих NFT [18], оракули) та висока ліквідність активів на децентралізованих біржах та NFT-маркетплейсах (OpenSea, Blur).

Узагальнюючи результати проведеного аналізу, зроблено висновок, що хоча окремі компоненти — TMA, P&E-моделі та ігрові блокчейни — активно розвиваються, їх комплексна інтеграція для створення графічно складної Action RPG у середовищі Telegram є малодослідженою областю. Існує чіткий науково-технічний "розрив" та ринкова ніша для створення платформи, яка б поєднала низький поріг входу та віральність (від TMA), глибокий ігровий процес (від Action RPG), стійку економіку (від P&E) та безпеку володіння активами (від блокчейну). Проведене дослідження підтвердило високу актуальність обраної теми та сформувало необхідний теоретичний базис для проєктування та реалізації платформи.

1.2 Аналіз програмних продуктів та аналогів

Для визначення ринкового позиціонування та конкурентних переваг гейміфікованої платформи, що розроблялася в рамках даної кваліфікаційної роботи, проведено детальний аналіз існуючих програмних продуктів-аналогів. Аналіз сфокусовано на проектах, що представляють різні аспекти ринку: домінуючі ігри на платформі Telegram Mini Apps, що демонструють поточні тренди; складні P2E-ігри жанру Action RPG на інших платформах, що встановлюють стандарт якості геймплею; та знакові проекти, що ілюструють ключові особливості GameFi економік.

Hamster Kombat

- Платформа: Telegram Mini App.
- Жанр: Симулятор управління криптобіржею у форматі «клікера» з елементами Idle-гри.
- P2E-модель: Проект використовує модель "Play-to-Airdrop", де гравці накопичують внутрішньоігрову валюту з очікуванням її майбутньої конвертації в реальний токен на блокчейні TON.
- Блокчейн: The Open Network (TON).
- Графіка: Проста 2D-графіка з мультяшною стилістикою, оптимізована для швидкого завантаження.
- Аналіз: Ключова перевага Hamster Kombat — максимальне використання вірального потенціалу Telegram. Реферальна система та прості механіки дозволили залучити десятки мільйонів користувачів. Головним недоліком є надзвичайно примітивний ігровий процес, що може призвести до швидкої втрати інтересу аудиторії після завершення фази ейрдропу.

Catizen

- Платформа: Telegram Mini App.
- Жанр: Казуальна гра з механікою «Merge» та елементами Idle-гри.
- P2E-модель: Аналогічна до Hamster Kombat модель "Play-to-Airdrop".
- Блокчейн: The Open Network (TON).
- Графіка: Яскрава та приваблива 2D-графіка в мультяшному стилі.
- Аналіз: Catizen пропонує дещо складніший ігровий цикл, ніж класичний «клікер», що потенційно забезпечує краще утримання гравців. Однак, як і попередній аналог, він страждає від поверхневості геймплею та залежності від майбутнього ейрдропу.

Illuvium

- Платформа: ПК (клієнт для завантаження на Windows).
- Жанр: Екосистема з кількох ігор, що включає Auto-battler та майбутню Action RPG.

- P2E-модель: Комплексна економіка, побудована навколо NFT-істот (Illuvials). Використовується токен $ILV для стейкінгу та управління.
- Блокчейн: Ethereum та Immutable X (Layer 2).
- Графіка: Високоякісна 3D-графіка на рушії Unreal Engine 5.
- Аналіз: Illuvium є прикладом амбітного Web3-проекту, що ставить на перше місце якість. Ключовим недоліком є надзвичайно високий поріг входу: необхідність мати потужний ПК, встановлювати крипто-гаманець та розбиратися у складній економіці.

Big Time

- Платформа: ПК (клієнт для завантаження на Windows).
- Жанр: Багатокористувацька Action RPG.
- P2E-модель: Інноваційна модель, орієнтована на заробіток косметичних NFT-предметів, які не впливають на ігровий баланс.
- Блокчейн: Власна пропрієтарна блокчейн-інфраструктура.
- Графіка: Стилізована 3D-графіка на рушії Unreal Engine.
- Аналіз: Big Time є одним з найближчих жанрових аналогів. Його перевага — динамічний геймплей та стійка економічна модель.
- Недоліки аналогічні до Illuvium: високий поріг входу та прив'язка до ПК.

Для наочного порівняння ключові характеристики проаналізованих продуктів та цільові показники гейміфікованої платформи, що розробляється, зведено у таблицю 1.

Жанр	Клікер / Idle	Merge / Idle	Auto-battler / RPG	Action RPG	Action RPG
P2E-модель	Play-to-Airdrop	Play-to-Airdrop	Комплексна (NFT, токени)	Косметичні NFT	Прямий обмін ігрових ресурсів на криптовалюту (TON)

Блокчейн	TON	TON	Ethereum / Immutable X	Пропрієтарний	TON (основний), потенційно BNB Chain
Графіка	Проста 2D	Проста 2D	Високоякісна 3D	Стилізована 3D	Комбінована 2D/3D, оптимізована для мобільних
Переваги	Максимальна віральність, низький поріг входу.	Захоплююча казуальна механіка, низький поріг входу.	Глибина геймплею, висока якість графіки.	Якісний A-RPG геймплей, стійка економічна модель.	Низький поріг входу, віральність, глибина A-RPG, пряма P2E-модель.
Недоліки	Дуже простий геймплей, відсутність довгострокової цінності.	Поверхневий геймплей, залежність від ейрдропу.	Дуже високий поріг входу, вимоги до «заліза».	Високий поріг входу (тільки ПК), нішева аудиторія.	Технічні виклики реалізації складної гри в TMA, ризики балансування економіки.

Таблиця 1. Ключові характеристики проаналізованих продуктів.

Джерело: запропоноване автором

Проведений аналіз програмних продуктів-аналогів дозволив зробити кілька ключових висновків.

По-перше, на ринку виявлено чітку дихотомію. З одного боку, існують надзвичайно успішні проекти на платформі Telegram Mini Apps (Hamster Kombat, Catizen), які демонструють колосальний потенціал для залучення мільйонної аудиторії завдяки низькому порогу входу. Однак їхньою слабкою стороною є примітивність ігрового процесу.

З іншого боку, існують високоякісні Web3-ігри жанру Action RPG (Illuvium, Big Time), які пропонують глибокий, захоплюючий геймплей. Проте вони орієнтовані на вузький сегмент "кор"-геймерів та мають надзвичайно високий поріг входу, що вимагає від користувачів потужного апаратного забезпечення та глибоких знань у сфері криптовалют.

Таким чином, встановлено наявність значної незайнятої ринкової ніші для продукту, який би зміг поєднати найкраще з обох світів: масову доступність платформи Telegram Mini Apps з глибиною та реіграбельністю жанру Action RPG.

Гейміфікована платформа, розроблена в рамках даної роботи, була стратегічно спозиціонована саме для заповнення цієї прогалини. Вона спрямована на те, щоб запропонувати мільйонам користувачів Telegram значно глибший ігровий досвід, ніж існуючі «клікери», водночас усуваючи технічні та фінансові бар'єри, властиві традиційним Web3-іграм. Це обґрунтовує вибір її архітектурних та геймдизайнерських рішень.

1.3 Постановка задачі

На основі проведеного в попередніх підрозділах глибокого аналізу предметної області, що включав огляд останніх науково-технічних досліджень у сферах гейміфікації та блокчейн-технологій, а також детального конкурентного аналізу існуючих програмних продуктів-аналогів, було сформульовано комплексну постановку задачі для даної кваліфікаційної роботи. Проведений аналіз однозначно підтвердив наявність значної та вільної ринкової ніші для інноваційного продукту, що здатен синергетично поєднати масову доступність та низький поріг входу платформи Telegram Mini Apps з глибиною та залученістю ігрового процесу, характерного для жанру Action RPG, та доповнити це економічними стимулами прозорої моделі Play-to-Earn. Таким чином, ключова задача полягала у розробці комплексної програмної системи, що повною мірою відповідає на цей ринковий запит, та функціональність якої чітко визначається набором детальних функціональних і нефункціональних вимог.

Основною стратегічною метою було визначено архітектурне проєктування, розробку та подальше впровадження архітектурно стійкої, надійної та високомасштабованої гейміфікованої платформи на базі екосистеми Telegram з іманентною інтеграцією технології блокчейн. Передбачалося, що розроблена платформа надасть кінцевим користувачам унікальний ігровий досвід у популярному жанрі Action RPG та інтегрує в ядро геймплею прозору та зрозумілу P2E-модель. Ця модель дозволить гравцям конвертувати ігрові досягнення та витрачений час в реальну економічну цінність через механізм обміну внутрішньоігрових токенізованих ресурсів на нативну криптовалюту мережі TON (The Open Network).

Функціональні вимоги, що визначають увесь спектр можливостей системи, для зручності аналізу та проєктування було згруповано за кількома логічними функціональними блоками.

Вимоги до ігрового процесу передбачали ретельну реалізацію ключових механік, що є фундаментальними для жанру Action RPG. До них належать: пряме керування персонажем у реальному часі для забезпечення максимального занурення; динамічна бойова система, що включає комбінацію базових атак та унікальних навичок з часом відновлення; багатогранна система прогресії персонажа (підвищення рівнів, розподіл очок характеристик); розгалужена квестова система як основний інструмент наративного просування; а також комплексна система здобичі (Loot & Items) з процедурною генерацією предметів різної рідкісності та характеристик.

Вимоги до P2E-економіки та блокчейн-інтеграції фокусувалися на створенні стійкої внутрішньоігрової економічної моделі. Вони включали впровадження основного внутрішньоігрового ресурсу (утилітарного токена), який гравці заробляють шляхом безпосередньої ігрової активності. Було визначено критичну необхідність забезпечення безшовної інтеграції з крипто-гаманцями мережі TON для мінімізації будь-яких перешкод для користувача. Також вимагалося реалізувати надійний та прозорий механізм обміну зароблено ресурсу на

криптовалюту та забезпечити повну прозорість усіх P2E-транзакцій шляхом їх фіксації в публічному реєстрі блокчейну, що гарантує довіру до системи.

Вимоги до платформи Telegram Mini App полягали в максимальному використанні переваг даної екосистеми. Ключовим було забезпечення автоматичної авторизації користувача на основі його облікового запису Telegram для безперешкодного входу в гру. Також вимагалося створення повністю адаптивного та відзивчивого користувацького інтерфейсу, оптимізованого для широкого спектра мобільних пристроїв. Окремий акцент робився на реалізації механізмів віральної соціальної взаємодії, як-от запрошення друзів за винагороду, що є рушієм органічного зростання аудиторії.

Вимоги до адміністративної панелі передбачали створення комплексного інструментарію для операційного управління платформою. Це включало модулі для управління користувацькими акаунтами та ігровим контентом в реальному часі (створення та редагування квестів, предметів, NPC), а також наявність аналітичних дашбордів. Ці дашборди мали надавати змогу моніторити ключові показники P2E-економіки (співвідношення емісії та спалювання токенів, швидкість обігу) з метою її оперативного балансування та запобігання інфляції.

Окрім функціональних, було сформульовано низку критичних нефункціональних вимог до якісних атрибутів системи, що є запорукою її довгострокового успіху.

— Продуктивність: Ігровий клієнт повинен був забезпечувати стабільну та плавну частоту кадрів (не нижче 30 FPS) на цільових мобільних пристроях для комфортного геймплею. Час відгуку серверної частини на дії користувача — не перевищувати порогового значення у 200 мс, що є критичним для ігор в реальному часі.

— Надійність: Система мала бути спроєктована для роботи в безперервному режимі 24/7 з показником доступності не нижче 99.5%. Обов'язковою була

реалізація механізмів автоматичного резервного копіювання баз даних для запобігання втраті даних користувачів.

— Масштабованість: Архітектура серверної частини була свідомо спроектована для забезпечення горизонтального масштабування. Це дозволяє гнучко нарощувати обчислювальні потужності для підтримки зростаючої кількості одночасних гравців, з цільовим показником у понад 100,000.

— Безпека: Вимоги включали імплементацію комплексних заходів для захисту від поширених веб-атак (XSS, SQL Injection, CSRF), наскрізне шифрування даних користувачів при зберіганні та передачі, та обов'язковий незалежний аудит безпеки смарт-контрактів, що реалізують P2E-логіку, для мінімізації фінансових ризиків.

— Сумісність: Платформа мала гарантовано коректно функціонувати на останніх стабільних версіях операційних систем iOS та Android в рамках офіційних клієнтів месенджера Telegram.

Таким чином, сформульований деталізований набір вимог став вичерпним технічним завданням, що слугувало фундаментальною основою та чіткою дорожньою картою для наступних етапів роботи — архітектурного проєктування та безпосередньої програмної реалізації даної комплексної гейміфікованої платформи.

РОЗДІЛ 2

МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ПЛАТФОРМИ

Після всебічного аналізу предметної області та визначення детальних вимог до системи в попередньому розділі, наступним логічним етапом є перехід від визначення того, *що* система повинна робити, до проектування того, *як* вона буде структурована та функціонувати. Цей розділ присвячений розробці архітектурних та проектних рішень, які є "кресленням" майбутньої гейміфікованої платформи.

Метою цього етапу є створення чітких, формалізованих моделей та схем, які слугуватимуть основою для безпосередньої програмної реалізації. Проектування включатиме моделювання функціональності системи за допомогою мови UML, розробку загальної архітектури інформаційної системи та детальне проектування моделі бази даних. Такий підхід дозволяє мінімізувати ризики, пов'язані з невірним тлумаченням вимог, та забезпечити створення надійної, масштабованої та підтримуваної системи.

2.1 Моделювання платформи

Для візуалізації, специфікації та документування архітектури розробленої гейміфікованої платформи було застосовано уніфіковану мову моделювання (UML). Побудовано моделі, що описують систему зі статичного ракурсу (функції, доступні користувачам) та динамічного (послідовність виконання ключових процесів).

Для моделювання функціональних вимог системи було побудовано діаграму варіантів використання. На основі вимог, визначених у попередньому розділі, ідентифіковано два ключові актори: Гравець (основний користувач, що взаємодіє з ігровим клієнтом) та Адміністратор (уповноважена особа, відповідальна за

управління системою через адміністративну панель). Загальна структура взаємодії акторів із системою представлена на діаграмі(Мал. 2.1).

Малюнок 2.1 Діаграма варіантів використання системи.

Джерело: запропоноване автором

Діаграма візуалізує наступні ключові функції:

Для «Гравець» передбачено функції:

- Авторизація через Telegram;
- Керування персонажем у ігровому світі;
- Участь у бойовій системі;
- Виконання квестів;
- Збір предметів та ресурсів («осколків»);
- Керування екіпіруванням;
- Прогресія персонажа (підвищення рівня, вивчення навичок);
- Підключення TON-гаманця;
- Обмін внутрішньоігрових ресурсів на криптовалюту TON.

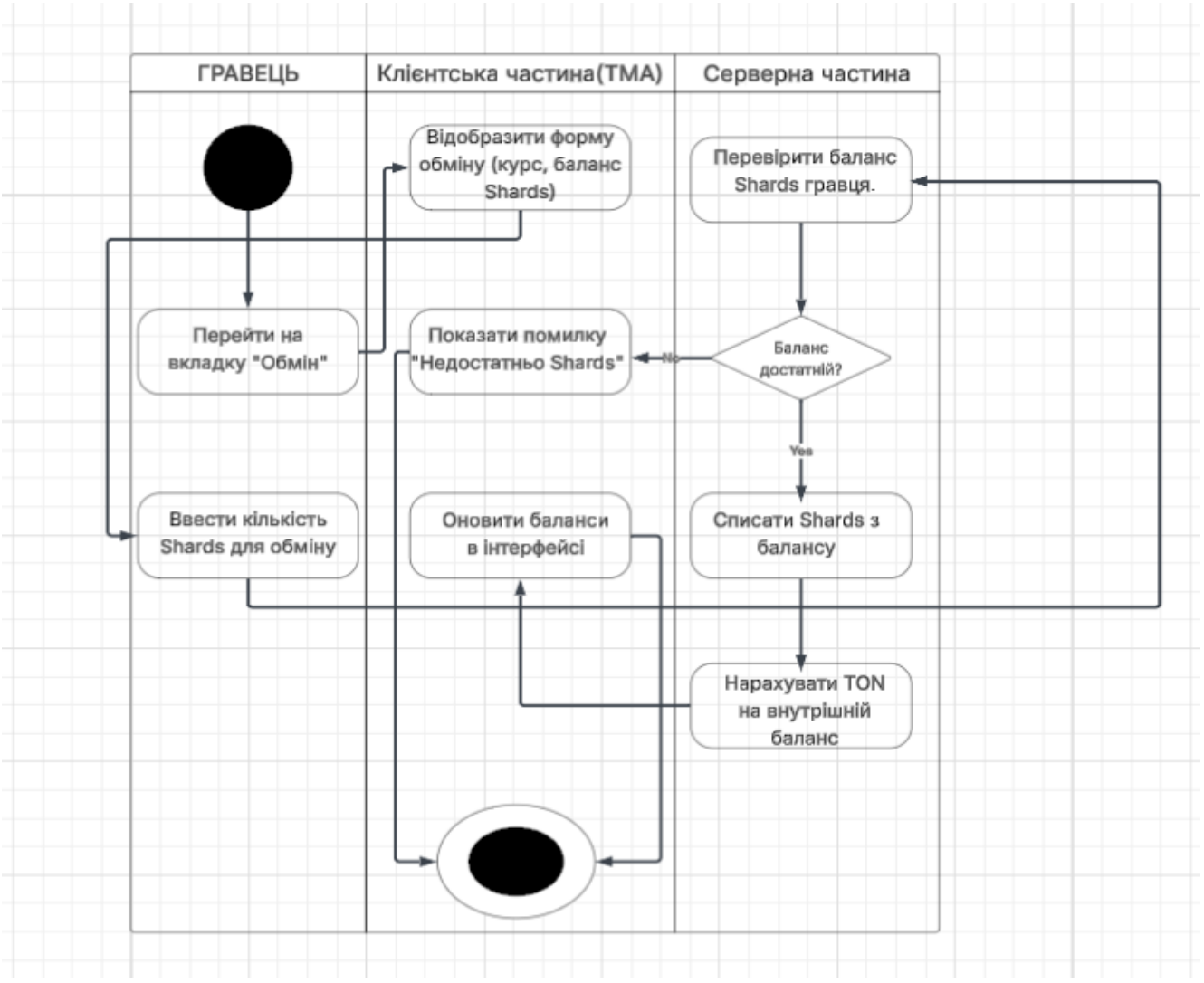
Для «Адміністратор» передбачено функції:

- Авторизація в адміністративній панелі;

- Керування ігровим контентом (квести, предмети, вороги);
- Перегляд статистики гравців;
- Моніторинг показників P2E-економіки.

Для моделювання динамічної поведінки та деталізації логіки виконання складних операцій було побудовано діаграми діяльності. Для детального аналізу обрано ключовий процес, що реалізує P2E-механіку. З метою оптимізації витрат на комісії в блокчейні та покращення користувацького досвіду, процес було розділено на два логічні етапи: внутрішній обмін ресурсів (off-chain) та виведення коштів на гаманець гравця (on-chain).

Перший етап — внутрішній обмін ігрових ресурсів. Цей процес відбувається виключно на сервері та в базі даних платформи (off-chain), що забезпечує миттєве виконання та відсутність комісій. Покрокова логіка процесу представлена на діаграмі(Мал 2.2).

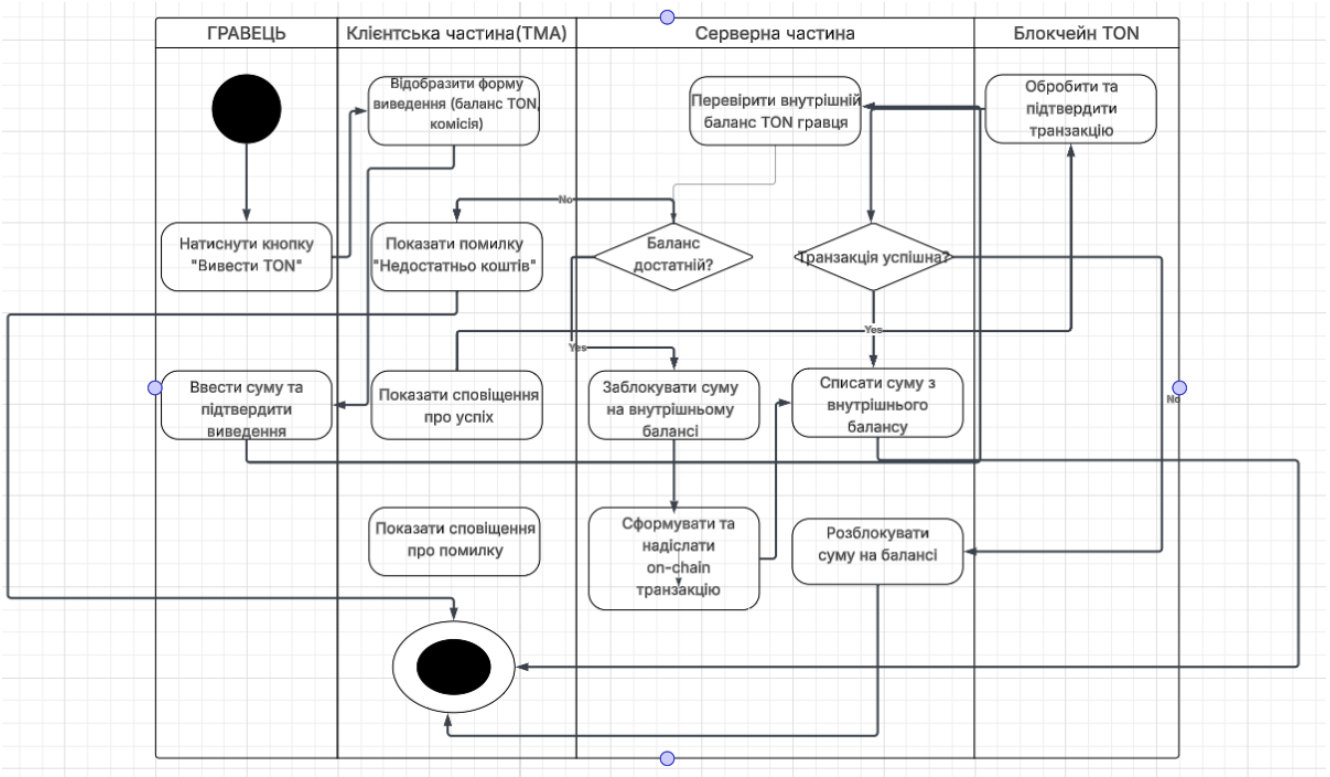


Малюнок 2.2 Діаграма діяльності процесу внутрішнього обміну «Shards»

Джерело: запропоноване автором

Процес ініціюється Гравцем через ігровий інтерфейс (TMA), який відображає форму обміну. Після введення бажаної суми, Серверна частина (Backend) виконує валідацію балансу. Якщо ресурсів достатньо, сервер проводить атомарну операцію: списує «осколки» та нараховує еквівалентну суму на внутрішній баланс гравця в TON. Після цього клієнтська частина оновлює інтерфейс користувача.

Другий етап — виведення коштів на гаманець. Ця операція є on-chain, оскільки передбачає реальну транзакцію в блокчейні. Логіка процесу детально показана на відповідній діаграмі.



Малюнок 2.3 Діаграма діяльності процесу виведення TON на гаманець

Джерело: запропоноване автором

Процес також ініціюється Гравцем. Після підтвердження суми Серверна частина перевіряє внутрішній баланс, тимчасово блокує суму в базі даних та

формує on-chain транзакцію, відправляючи її на обробку до Блокчейну TON. Якщо транзакція в мережі проходить успішно, сервер остаточно списує суму з внутрішнього балансу гравця. Якщо блокчейн повертає помилку, сума розблоковується і повертається на внутрішній баланс. Клієнтська частина інформує користувача про фінальний результат операції.

Таким чином, представлене моделювання забезпечило чітку візуалізацію та специфікацію функціональної й динамічної поведінки системи, що стало основою для подальшої технічної реалізації.

2.2. Проектування інформаційної системи

Після етапу високорівневого моделювання було виконано проектування конкретної технічної архітектури гейміфікованої платформи. Правильно спроектована архітектура є фундаментальною передумовою для створення надійної, масштабованої та легко підтримуваної системи. Для розробки платформи було обрано класичну трирівневу клієнт-серверну архітектуру. Ця модель забезпечує чітке розмежування відповідальності між компонентами системи, що значно спрощує процеси розробки, тестування та подальшого розвитку.

Архітектура системи складається з наступних трьох логічних рівнів:

Рівень представлення (Presentation Tier). Це клієнтська частина, що функціонує в середовищі Telegram Mini App (TMA) на пристрої користувача. Вона відповідає за візуалізацію ігрового світу, користувацького інтерфейсу, обробку дій гравця (введення, рух) та комунікацію з сервером. Враховуючи обмеження середовища WebView, в якому працюють TMA, технологічний стек було підібрано для досягнення балансу між якістю графіки та продуктивністю. Основною мовою програмування є JavaScript. Для рендерингу 2D-інтерфейсів, ігрових меню та спецефектів використано бібліотеку Pixi.js, що відома своєю високою продуктивністю. Для рендерингу 3D-моделей персонажів та ключових об'єктів оточення застосовано бібліотеку Three.js. Такий комбінований підхід дозволив створити візуально привабливий продукт, оптимізований для мобільних пристроїв.

Рівень логіки (Logic/Application Tier). Це серверна частина (Backend), яка є центральним вузлом та "мозком" усієї системи. Вона виступає як авторитетне джерело істини для запобігання шахрайству та є відповідальною за обробку всієї бізнес-логіки гри: правил бою, логіки квестів, управління станом гравців та їх авторизації. Також цей рівень забезпечує безпечну взаємодію з базою даних та блокчейном. Основною мовою програмування обрано Python через його багату екосистему. Для реалізації основного ігрового API було використано асинхронний фреймворк FastAPI, що забезпечує високу пропускну здатність та низькі затримки, критичні для ігрових застосунків. Водночас для адміністративної панелі було застосовано фреймворк Django, який надає надійний та безпечний функціонал для управління контентом та моніторингу системи "з коробки".

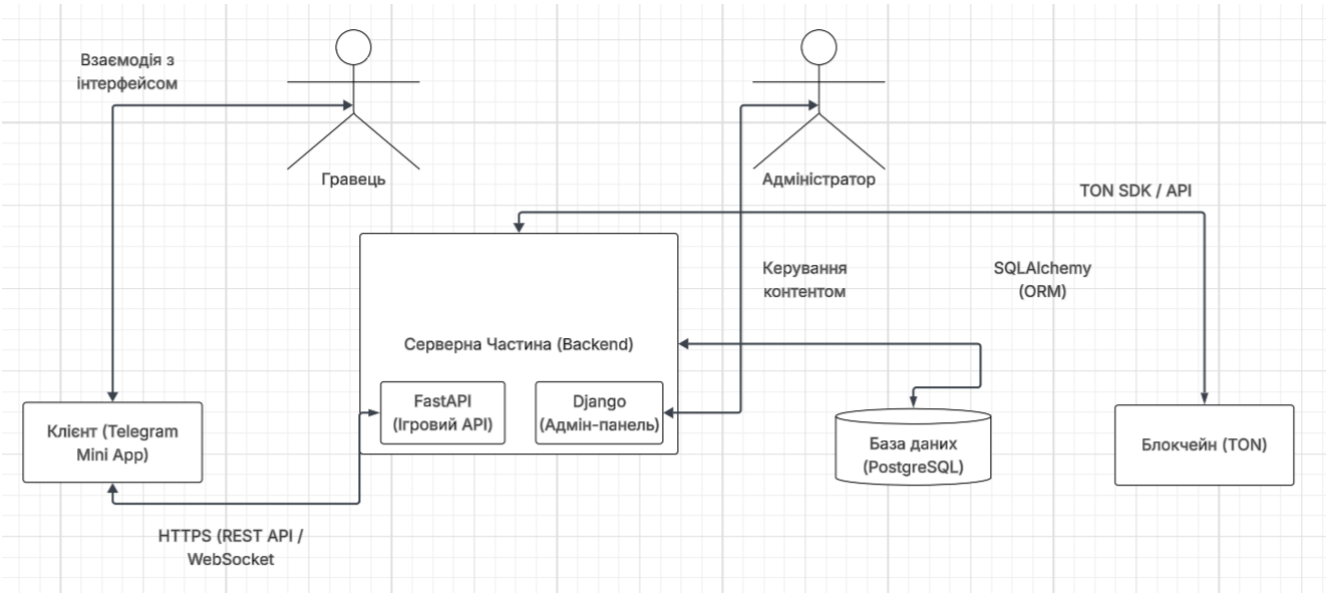
Рівень даних (Data Tier). Цей рівень складається з двох основних сховищ, що виконують різні завдання.

Реляційна база даних. Відповідає за надійне та структуроване зберігання всіх ігрових даних: інформації про гравців, їхній прогрес, предмети, внутрішні баланси ресурсів тощо. Для цих цілей було обрано систему управління базами даних PostgreSQL, відому своєю надійністю, відповідністю принципам ACID та розширюваністю. Взаємодія з нею з боку серверної частини відбувається через ORM (Object-Relational Mapper) SQLAlchemy.

Блокчейн. Використовується як децентралізований реєстр для обліку та передачі цінних P2E-активів. Це забезпечує прозорість та гарантує гравцям справжнє право власності на зароблені кошти. Для проекту було обрано блокчейн The Open Network (TON) через його глибоку інтеграцію з екосистемою Telegram. Сервер взаємодіє з вузлами мережі TON через відповідні SDK-бібліотеки для мови Python.

Взаємодія між клієнтською та серверною частинами відбувається переважно через RESTful API за захищеним протоколом HTTPS. Для ігрових подій, що вимагають обміну даними в реальному часі (наприклад, бій), додатково використовується протокол WebSocket для забезпечення мінімальної затримки.

Загальна архітектура системи та взаємодія її компонентів наочно представлена на схемі(Мал. 2.4).



Малюнок 2.4 Архітектурна схема гейміфікованої платформи

Джерело: запропоноване автором

Представлена на схемі архітектура візуалізує потік даних та взаємодію між компонентами. Користувач (Гравець) взаємодіє з клієнтською частиною (TMA), яка через захищений API-шлюз відправляє запити до серверної частини (Backend). Backend, в свою чергу, обробляє ігрову логіку, звертаючись до бази даних PostgreSQL для отримання та збереження стану гравця. Для виконання P2E-операцій (обмін та виведення коштів), сервер комунікує з вузлами блокчейну TON. Адміністратор має окремий, захищений доступ до серверної частини через адміністративну панель Django для управління контентом та моніторингу системи. Така чітка структуризація дозволяє забезпечити надійність, безпеку та масштабованість платформи.

2.3 Проектування моделі бази даних

Наступним кроком після визначення загальної архітектури стало проектування структури сховища даних. Правильна організація даних є критично важливою для ігрової платформи з персистентним світом та складною економікою.

Метою даного етапу було створення логічної та фізичної моделі даних, яка є ефективною, масштабованою та здатною підтримувати всі раніше визначені функціональні вимоги.

Як було обґрунтовано в архітектурі системи, в якості системи управління базами даних (СУБД) було обрано PostgreSQL. Цей вибір зумовлено кількома ключовими перевагами. По-перше, надійність та відповідність принципам ACID забезпечують повну підтримку транзакцій, що є критично важливим для цілісності фінансових операцій (обмін, виведення) та запобігання втраті прогресу гравців через можливі збої. По-друге, доведена масштабованість PostgreSQL дозволяє ефективно працювати з великими обсягами даних та високими навантаженнями. По-третє, розширюваність та гнучкість, зокрема підтримка типу даних JSONB, надає значні переваги. Це дозволило зберігати складні, напівструктуровані дані, як-от атрибути предметів чи характеристики персонажів, в одному полі, що спрощує подальше розширення ігрової логіки без необхідності виконувати складні міграції схеми даних.

Проектування бази даних здійснено на основі ER-моделі (Entity-Relationship Model). Було спроектовано логічну структуру, що складається з набору ключових сутностей (таблиць).

Таблиця users. Ця таблиця є центральною для даних користувача, ідентифікованих за їх унікальним ідентифікатором в Telegram. Розділення даних акаунта та ігрового персонажа дозволяє в майбутньому реалізувати створення кількох персонажів на один акаунт. Ключові атрибути: telegram_id (унікальний ідентифікатор з Telegram, використовується для авторизації), internal_ton_balance (внутрішній баланс TON, тип Decimal використано для запобігання помилок округлення, властивих типам з плаваючою комою), wallet_address (адреса підключеного TON-гаманця, може бути NULL).

Таблиця characters. Зберігає всю інформацію, що стосується безпосередньо ігрових персонажів та їхнього прогресу. Атрибути: level (рівень), experience (досвід), shards_balance (баланс основної ігрової валюти, тип BigInt обрано з

розрахунком на великі числові значення), stats (характеристики персонажа типу JSONB, що дозволяє легко додавати нові, наприклад, "опір до магії", без зміни структури таблиці).

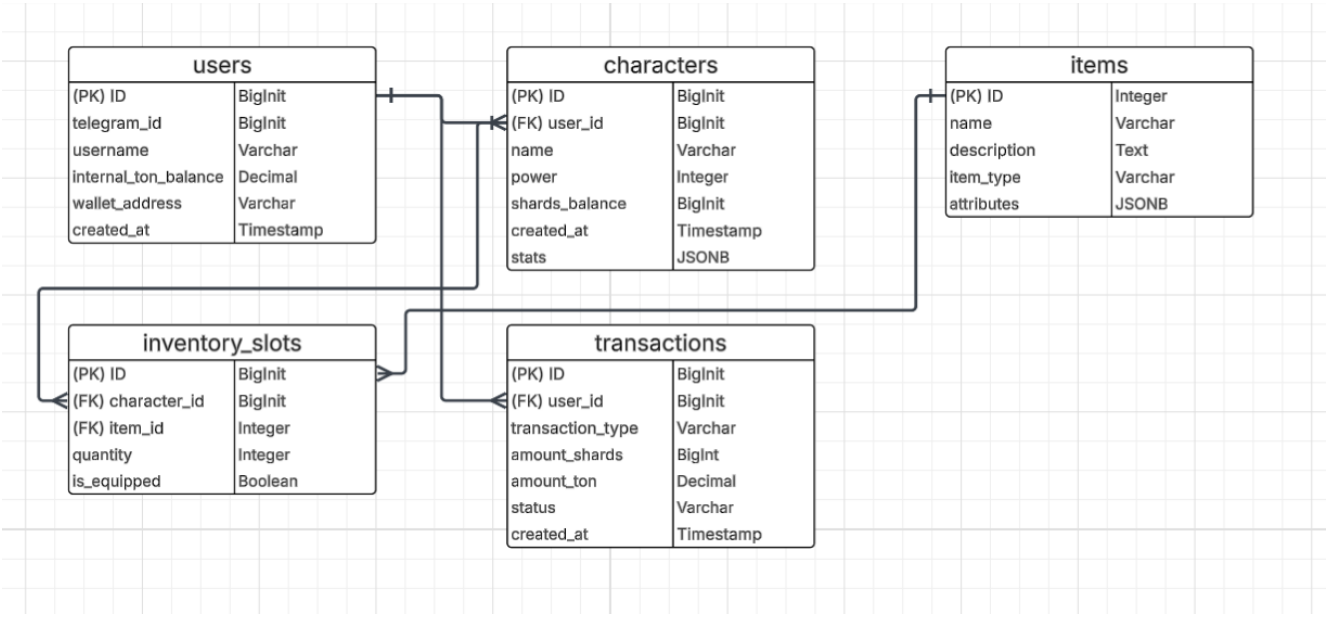
Таблиця items. Функціонує як довідник або "шаблон" усіх можливих предметів у грі. Вона містить статичну інформацію: name (назва), description (опис), item_type (категорія, напр., 'WEAPON', 'ARMOR'), attributes (бонуси, які надає предмет, у форматі JSONB). Такий підхід відокремлює незмінні дані про предмети від екземплярів, якими володіють гравці.

Таблиця inventory_slots. Це зв'язуюча таблиця, що реалізує відношення "багато-до-багатьох" між персонажами та предметами. Кожен запис означає, що певний персонаж (character_id) володіє певним предметом (item_id) у певній кількості (quantity). Атрибут is_equipped (прапорець) є важливою оптимізацією, що дозволяє швидко вибирати екіпіровані предмети для розрахунку характеристик персонажа без додаткових умов.

Таблиця transactions. Виконує роль журналу аудиту для всіх економічних операцій. Вона є критично важливою для аналізу економіки гри, вирішення спірних питань з користувачами та відстеження фінансових потоків. Таблиця зберігає transaction_type (тип операції), amount_shards (кількість "осколків"), amount_ton (кількість TON) та status ('SUCCESS', 'PENDING', 'FAILED'), що необхідно для відстеження асинхронних on-chain операцій.

Між визначеними таблицями існують чіткі реляційні зв'язки, що забезпечують цілісність даних: зв'язок users ↔ characters типу "один-до-багатьох"; зв'язок characters ↔ inventory_slots та items ↔ inventory_slots (разом утворюють зв'язок "багато-до-багатьох"); та зв'язок users ↔ transactions для зберігання історії операцій.

Візуальним представленням цієї логічної моделі є ER-діаграма(Мал. 2.5).



Малюнок 2.5: ER-діаграма бази даних платформи

Джерело: запропоноване автором

Представлена на рисунку ER-діаграма графічно зображує описані сутності, їхні атрибути та зв'язки. Вона візуально підтверджує, як таблиця users є кореневою для characters та transactions, а inventory_slots ефективно пов'язує characters та довідник items. Така структура забезпечує високий рівень нормалізації, цілісність даних за допомогою зовнішніх ключів та гнучкість, необхідну для подальшого розвитку та розширення ігрової платформи.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ГЕЙМІФІКОВАНОЇ ПЛАТФОРМИ НА БАЗІ TELEGRAM

3.1 Архітектура програмного додатку

Якщо архітектура інформаційної системи, розглянута в підрозділі 2.2, є генеральним планом міста, то архітектура програмного додатку — це детальний план забудови конкретного кварталу, що включає схеми комунікацій, поверхові плани та інженерні мережі. Вона визначає внутрішню організацію вихідного коду, його поділ на модулі, структуру директорій та файлів, а також правила взаємодії між цими модулями.

Вибір правильної архітектури коду є не менш важливим, ніж вибір технологій, оскільки хаотична та незапланована структура неминуче призводить до накопичення технічного боргу (technical debt). Це ускладнює подальшу підтримку, робить внесення змін повільним та ризикованим, а також підвищує поріг входження для нових розробників. Тому в основі архітектури даного проекту лежать фундаментальні принципи програмної інженерії: Розділення відповідальності (Separation of Concerns, SoC), Не повторюйся (Don't Repeat Yourself, DRY) та SOLID. Це дозволило створити не просто працюючий, а й гнучкий, підтримуваний та масштабований програмний продукт.

Структура та компоненти серверної частини (Backend):

Серверна частина, реалізована на мові Python, є центральним вузлом, що координує всю логіку платформи. Її архітектура побудована за модульним принципом з чітким розмежуванням функціональних шарів.

— /app/core - Шар конфігурації: Це "серце" та точка входу в налаштування додатку. Замість жорсткого кодування констант у коді, що є поганою практикою, всі налаштування (параметри підключення до БД,

секретні ключі, URL зовнішніх сервісів) зберігаються у файлі .env і зчитуються за допомогою бібліотеки Pydantic. Клас конфігурації, успадкований від BaseSettings, автоматично валідує наявність та тип змінних середовища при старті додатку, що запобігає цілому класу помилок, пов'язаних з невірною конфігурацією.

— /app/db - Шар доступу до даних (Data Access Layer): Цей модуль повністю інкапсулює логіку взаємодії з базою даних PostgreSQL.

— models.py: Тут декларативно описані всі класи-моделі SQLAlchemy, які є прямим відображенням ER-діаграми з підрозділу 2.3. Це забезпечує єдине "джерело правди" про структуру даних.

— session.py: Реалізує патерн "Unit of Work" та налаштовує механізм сесій SQLAlchemy. Завдяки системі впровадження залежностей (Dependency Injection) у FastAPI, кожна API-операція отримує власну сесію з БД, що гарантує ізольованість транзакцій.

— /app/api - Шар представлення API (API Presentation Layer): Директорія, де логіка FastAPI перетворює бізнес-логіку на доступні ззовні HTTP-Endpointи. Застосовано підхід з використанням APIRouter. Кожен файл (наприклад, characters_api.py, exchange_api.py) є логічним "контролером" для певної групи сутностей. Це дозволяє дотримуватися принципів RESTful і уникнути створення одного гігантського файлу main.py з усіма Endpointами, що значно покращує читабельність та підтримку коду. Валідація вхідних та вихідних даних на цьому рівні також здійснюється за допомогою моделей Pydantic, що гарантує цілісність даних на межі системи.

— /app/game_logic - Шар бізнес-логіки (Service Layer): Це найважливіший шар, куди винесена вся складна логіка, що не стосується безпосередньо обробки HTTP-запитів. Наприклад, функція розрахунку power персонажа, яка може залежати від його базових характеристик, бонусів від предметів та тимчасових ефектів, реалізована саме тут. Такий підхід (винесення логіки в "сервіси") дозволяє легко перевіряти її за допомогою unit-тестів та перевикористовувати в різних частинах додатку.

— /admin_panel - Адміністративний інтерфейс: Для реалізації адміністративної панелі було прийнято прагматичне рішення використати потужний вбудований функціонал фреймворку Django. Створення окремого Django-проекту, що працює з тією ж базою даних, дозволило за лічені години отримати безпечний, гнучкий та функціональний веб-інтерфейс для управління ігровим контентом (предметами, квестами) та перегляду даних користувачів, заощадивши сотні годин розробки кастомного рішення.

Структура та компоненти клієнтської частини (Frontend):

Клієнтська частина, реалізована на JavaScript, побудована за сцено-орієнтованим підходом, який є де-факто стандартом у розробці ігор. Цей підхід розглядає гру як скінченний автомат, де кожен стан — це окрема ігрова сцена.

— /src/scenes: Ключова директорія, що містить логіку окремих ігрових екранів. Кожна сцена (напр., MainMenuScene.js, GameplayScene.js, ExchangeScene.js) є класом, який інкапсулює об'єкти та логіку, що відображаються в даний момент. Кожна сцена має стандартний життєвий цикл: preload() (для завантаження ассетів, потрібних саме цій сцені), create() (для ініціалізації об'єктів та інтерфейсу) та update() (основний ігровий цикл, що викликається кожного кадру).

— /src/game_objects: Директорія для перевикористовуваних ігрових сутностей. Наприклад, тут може бути базовий клас GameObject, від якого успадковуються класи Player, Enemy, Projectile. Це дозволяє використовувати принципи об'єктно-орієнтованого програмування, такі як наслідування та поліморфізм, для уникнення дублювання коду.

— /src/services: Модулі, що реалізують патерн "Шлюз" (Gateway) для комунікації із зовнішнім світом.

— ApiService.js: Єдиний "комунікаційний хаб" для взаємодії з backend. Всі fetch-запити до ігрового API інкапсульовані тут. Це дозволяє легко керувати авторизацією (додавати токени до заголовків), обробляти помилки та, за потреби, кешувати відповіді в одному місці.

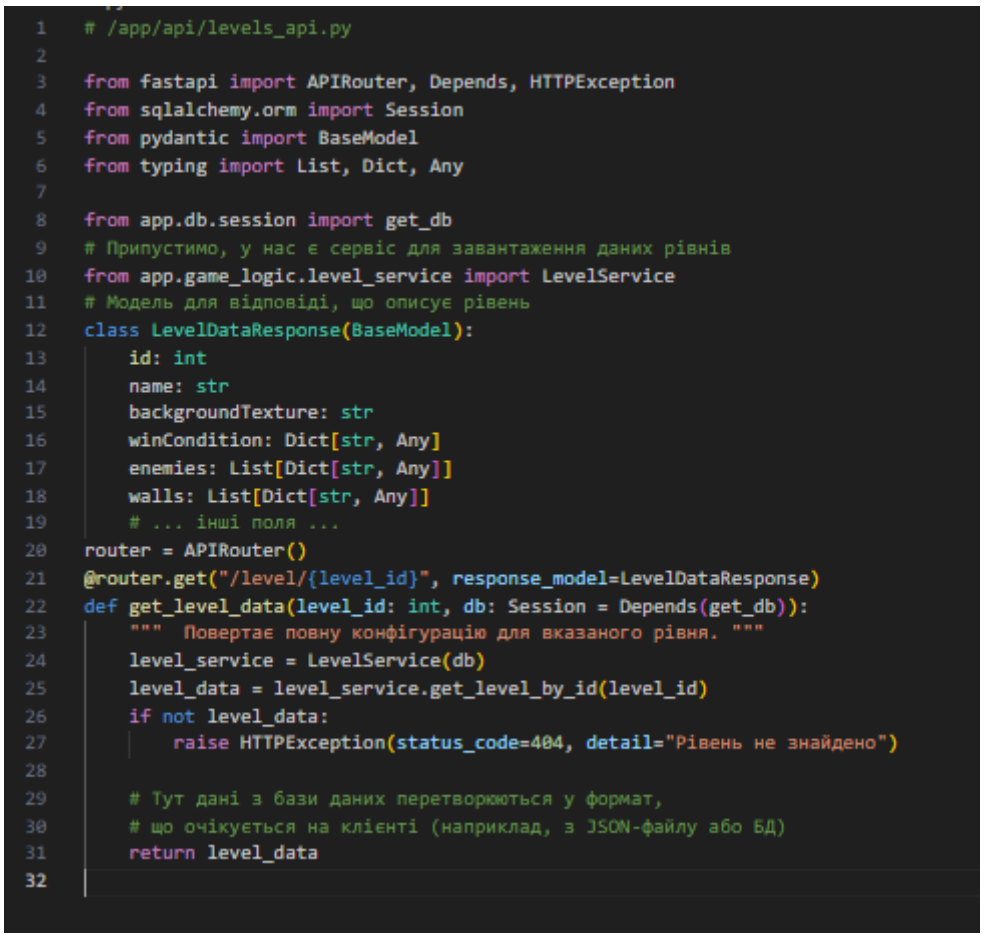
— WalletService.js: Інкапсулює логіку взаємодії з API TON-гаманця, надаючи простий інтерфейс для інших частин додатку (напр., connect(), sendTransaction()).

— /assets: Директорія для зберігання всіх статичних ресурсів — зображень (спрайтів, текстур), файлів 3D-моделей, звукових ефектів та музики.

На завершення, обрана архітектура програмного додатку не є випадковою, а є результатом свідомого застосування перевірених часом інженерних практик та дизайн-патернів. Вона створює систему, яка є не тільки функціональною на даний момент, але й закладає міцний фундамент для майбутнього розвитку, масштабування та підтримки.

3.2. Програмна реалізація

Оскільки серверна частина знаходиться на етапі проектування, наведемо концептуальний приклад її реалізації, що демонструє підхід до створення API для взаємодії з клієнтом. Однією з ключових функцій є надання клієнту даних про рівень та його сутності (ворогів). На малюнку 3.1 представлено фрагмент коду, що реалізує Endpoint для отримання конфігурації рівня.



```python
# /app/api/levels_api.py

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from pydantic import BaseModel
from typing import List, Dict, Any

from app.db.session import get_db
# Припустимо, у нас є сервіс для завантаження даних рівнів
from app.game_logic.level_service import LevelService
# Модель для відповіді, що описує рівень
class LevelDataResponse(BaseModel):
    id: int
    name: str
    backgroundTexture: str
    winCondition: Dict[str, Any]
    enemies: List[Dict[str, Any]]
    walls: List[Dict[str, Any]]
    # ... інші поля ...
router = APIRouter()
@router.get("/level/{level_id}", response_model=LevelDataResponse)
def get_level_data(level_id: int, db: Session = Depends(get_db)):
    """  Повертає повну конфігурацію для вказаного рівня. """
    level_service = LevelService(db)
    level_data = level_service.get_level_by_id(level_id)
    if not level_data:
        raise HTTPException(status_code=404, detail="Рівень не знайдено")

    # Тут дані з бази даних перетворюються у формат,
    # що очікується на клієнті (наприклад, з JSON-файлу або БД)
    return level_data

```

Малюнок 3.1 – Endpoint для отримання даних рівня

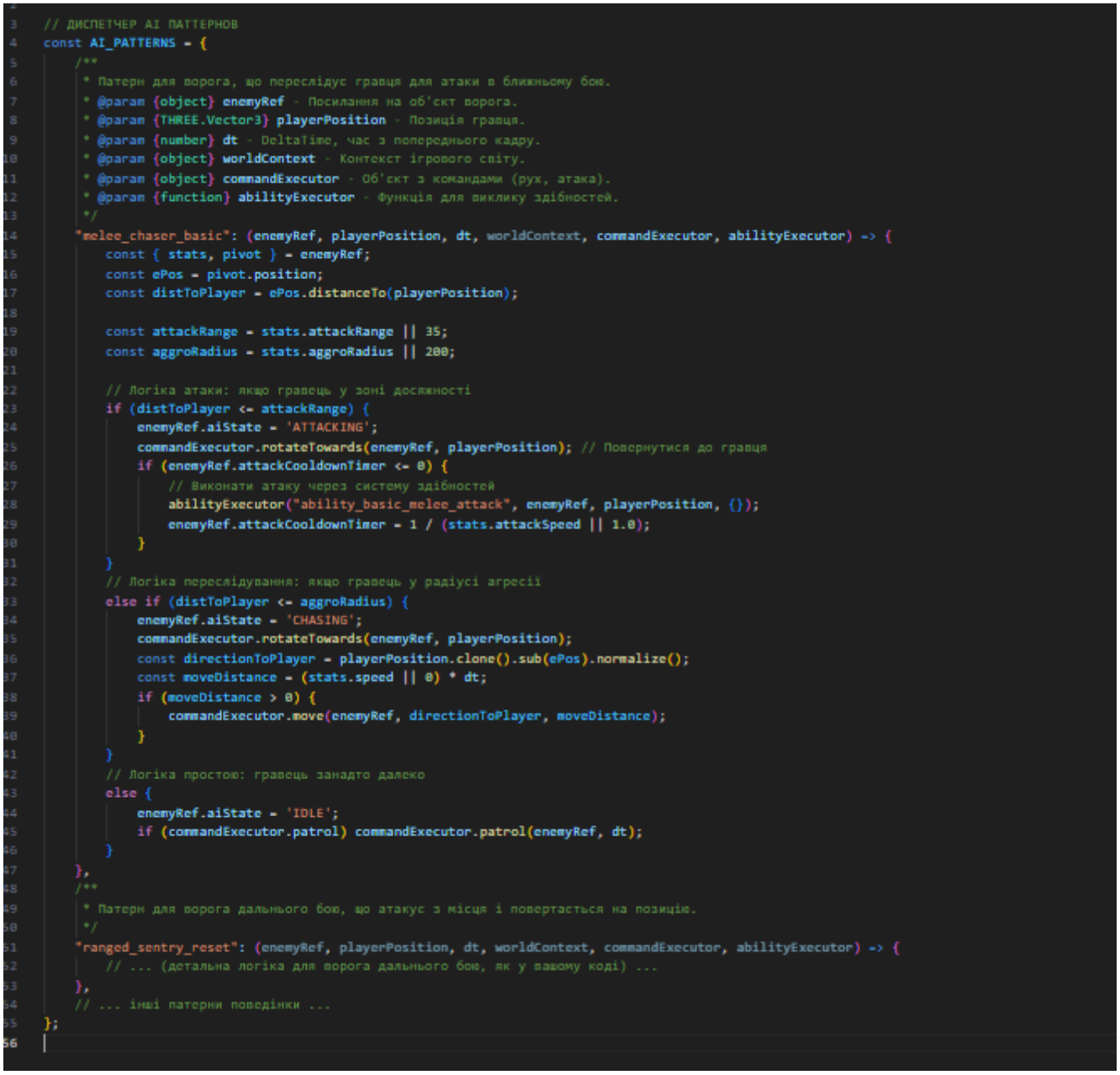
Джерело: запропоноване автором

Він приймає level_id як параметр шляху, використовує залежність get_db для отримання сесії бази даних та викликає спеціальний сервісний клас LevelService для отримання даних. Це ілюструє принцип розділення відповідальності: API-шар відповідає лише за прийом запитів та відправку відповідей, а вся логіка роботи з даними інкапсульована в сервісному шарі. Використання Pydantic-моделі LevelDataResponse гарантує, що відповідь сервера завжди матиме очікувану структуру.

Одним із найскладніших завдань на клієнтській стороні є реалізація поведінки неігрових персонажів (NPC), зокрема ворогів. Для цього було розроблено гнучку систему на основі патерну проектування **"Стратегія" (Strategy Pattern)**. Вся логіка поведінки винесена в окремий об'єкт-диспетчер

AI_PATTERNS, де кожна властивість — це функція, що реалізує конкретну модель поведінки.

На малюнку 3.2 представлено фрагмент цього диспетчера з реалізацією двох базов их патернів: для ворога ближнього бою та для ворога дальнього бою.



```
// ДИСПЕТЧЕР AI ПАТТЕРНОВ
const AI_PATTERNS = {
    /**
     * Патерн для ворога, що переслідує гравця для атаки в ближньому бою.
     * @param {object} enemyRef - Посилання на об'єкт ворога.
     * @param {THREE.Vector3} playerPosition - Позиція гравця.
     * @param {number} dt - DeltaTime, час з попереднього кадру.
     * @param {object} worldContext - Контекст ігрового світу.
     * @param {object} commandExecutor - Об'єкт з командами (рух, атака).
     * @param {function} abilityExecutor - Функція для виклику здібностей.
     */
    "melee_chaser_basic": (enemyRef, playerPosition, dt, worldContext, commandExecutor, abilityExecutor) => {
        const { stats, pivot } = enemyRef;
        const ePos = pivot.position;
        const distToPlayer = ePos.distanceTo(playerPosition);

        const attackRange = stats.attackRange || 35;
        const aggroRadius = stats.aggroRadius || 200;

        // Логіка атаки: якщо гравець у зоні досяжності
        if (distToPlayer <= attackRange) {
            enemyRef.aiState = 'ATTACKING';
            commandExecutor.rotateTowards(enemyRef, playerPosition); // Повернутися до гравця
            if (enemyRef.attackCooldownTimer <= 0) {
                // Виконати атаку через систему здібностей
                abilityExecutor("ability_basic_melee_attack", enemyRef, playerPosition, {});
                enemyRef.attackCooldownTimer = 1 / (stats.attackSpeed || 1.0);
            }
        }
        // Логіка переслідування: якщо гравець у радіусі агресії
        else if (distToPlayer <= aggroRadius) {
            enemyRef.aiState = 'CHASING';
            commandExecutor.rotateTowards(enemyRef, playerPosition);
            const directionToPlayer = playerPosition.clone().sub(ePos).normalize();
            const moveDistance = (stats.speed || 0) * dt;
            if (moveDistance > 0) {
                commandExecutor.move(enemyRef, directionToPlayer, moveDistance);
            }
        }
        // Логіка простою: гравець занадто далеко
        else {
            enemyRef.aiState = 'IDLE';
            if (commandExecutor.patrol) commandExecutor.patrol(enemyRef, dt);
        }
    },
    /**
     * Патерн для ворога дальнього бою, що атакує з місця і повертається на позицію.
     */
    "ranged_sentry_reset": (enemyRef, playerPosition, dt, worldContext, commandExecutor, abilityExecutor) => {
        // ... (детальна логіка для ворога дальнього бою, як у вашому коді) ...
    },
    // ... інші патерни поведінки ...
};
```

Малюнок 3.2 – Фрагмент диспетчера AI-патернів

Джерело: запропоноване автором

Даний підхід має кілька ключових переваг. По-перше, гнучкість та розширюваність: для додавання нового типу поведінки ворога достатньо написати

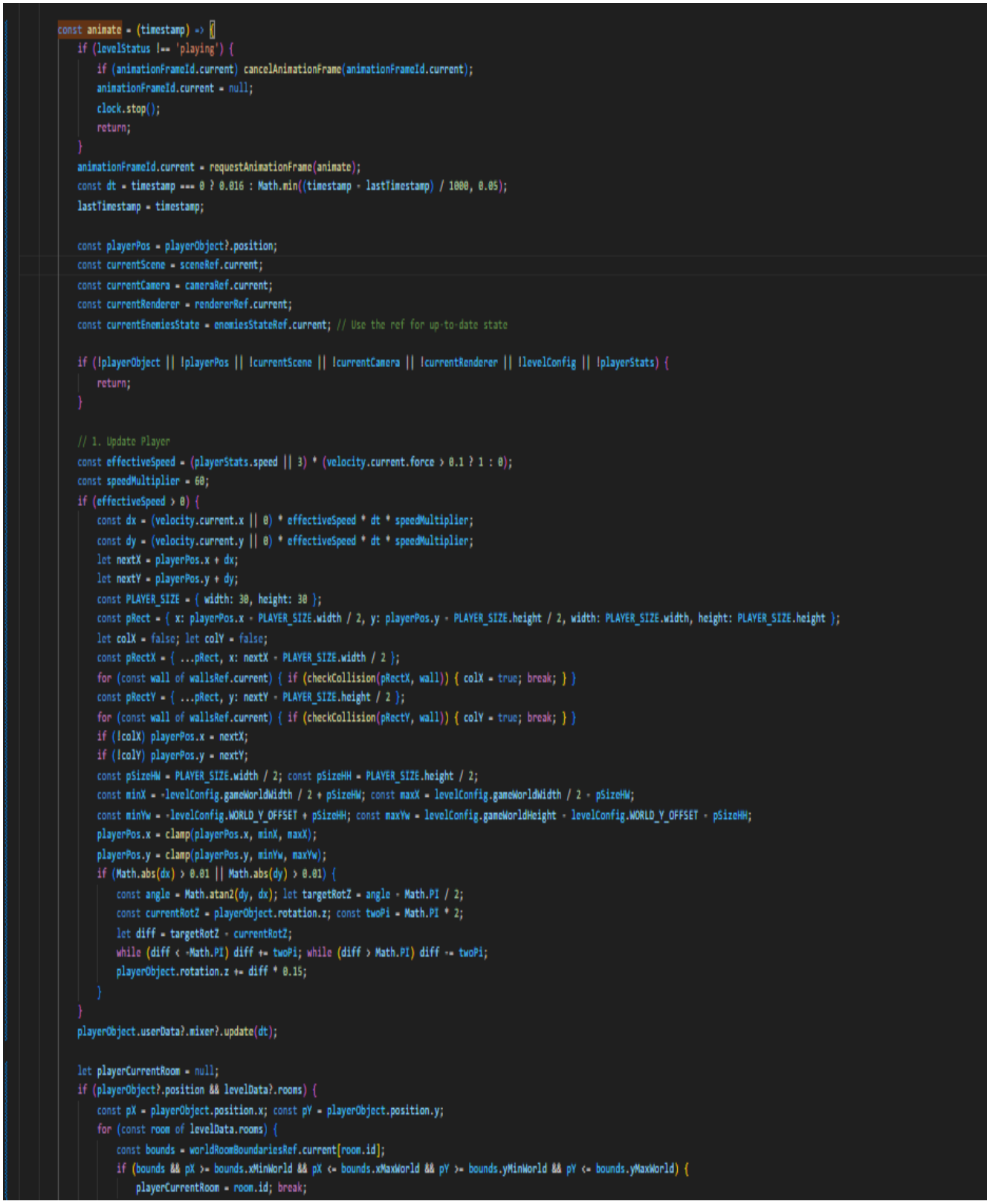
нову функцію і додати її в об'єкт AI_PATTERNS, не змінюючи основний ігровий цикл. По-друге, розділення відповідальності: ігровий цикл не містить складної логіки if/else для кожного типу ворога; він лише викликає відповідну функцію з диспетчера. По-третє, конфігурація через дані: тип поведінки для кожного ворога (melee_chaser_basic або ranged_sentry_reset) задається в конфігураційному файлі ENTITY_CONFIG(Мал. 3.6), що дозволяє геймдизайнеру змінювати поведінку ворогів, не втручаючись у програмний код. Це є прикладом ефективного та сучасного підходу до розробки ігрового штучного інтелекту.

Ігровий цикл є серцем будь-якої інтерактивної гри. В даному проекті він реалізований за допомогою хука useEffect в React та requestAnimationFrame для взаємодії з Three.js. На малюнку 3.3 показана спрощена структура ігрового циклу. На малюнку 3.4, 3.5 показана більше детальна структура окремих частин.

```
useEffect(() => {
    // ... (код ініціалізації сцени, камери, рендерера) ...

    const clock = new THREE.Clock();

    const animate = () => {
        // Запланувати виклик цієї ж функції на наступний кадр
        animationFrameId.current = requestAnimationFrame(animate);

        // Розрахунок delta time для плавності анімацій
        const dt = clock.getDelta();

        // 1. Оновлення гравця (рух, анімації)
        updatePlayer(dt);

        // 2. Оновлення ворогів (виконання AI)
        updateEnemies(dt);

        // 3. Оновлення снарядів та ефектів
        updateProjectiles(dt);

        // 4. Перевірка колізій
        checkCollisions();

        // 5. Оновлення камери
        updateCamera(dt);

        // 6. Рендеринг сцени
        renderer.render(scene, camera);
    };

    animate(); // Запускаємо ігровий цикл

    // Функція очищення при розмонтуванні компонента
    return () => {
        cancelAnimationFrame(animationFrameId.current);
        // ... інший код очищення ресурсів ...
    };
}, [/* залежності хука */]);
```

Малюнок 3.3 – Спрощена структура ігрового циклу

Джерело: запропоноване автором



```
const animate = (timestamp) => {
    if (levelStatus !== 'playing') {
        if (animationFrameId.current) cancelAnimationFrame(animationFrameId.current);
        animationFrameId.current = null;
        clock.stop();
        return;
    }
    animationFrameId.current = requestAnimationFrame(animate);
    const dt = timestamp === 0 ? 0.016 : Math.min((timestamp - lastTimestamp) / 1000, 0.05);
    lastTimestamp = timestamp;

    const playerPos = playerObject?.position;
    const currentScene = sceneRef.current;
    const currentCamera = cameraRef.current;
    const currentRenderer = rendererRef.current;
    const currentEnemiesState = enemiesStateRef.current; // Use the ref for up-to-date state

    if (!playerObject || !playerPos || !currentScene || !currentCamera || !currentRenderer || !levelConfig || !playerStats) {
        return;
    }

    // 1. Update Player
    const effectiveSpeed = (playerStats.speed || 3) * (velocity.current.force > 0.1 ? 1 : 0);
    const speedMultiplier = 60;
    if (effectiveSpeed > 0) {
        const dx = (velocity.current.x || 0) * effectiveSpeed * dt * speedMultiplier;
        const dy = (velocity.current.y || 0) * effectiveSpeed * dt * speedMultiplier;
        let nextX = playerPos.x + dx;
        let nextY = playerPos.y + dy;
        const PLAYER_SIZE = { width: 30, height: 30 };
        const pRect = { x: playerPos.x - PLAYER_SIZE.width / 2, y: playerPos.y - PLAYER_SIZE.height / 2, width: PLAYER_SIZE.width, height: PLAYER_SIZE.height };
        let colX = false; let colY = false;
        const pRectX = { ...pRect, x: nextX - PLAYER_SIZE.width / 2 };
        for (const wall of wallsRef.current) { if (checkCollision(pRectX, wall)) { colX = true; break; } }
        const pRectY = { ...pRect, y: nextY - PLAYER_SIZE.height / 2 };
        for (const wall of wallsRef.current) { if (checkCollision(pRectY, wall)) { colY = true; break; } }
        if (!colX) playerPos.x = nextX;
        if (!colY) playerPos.y = nextY;
        const pSizeHW = PLAYER_SIZE.width / 2; const pSizeHH = PLAYER_SIZE.height / 2;
        const minX = -levelConfig.gameWorldWidth / 2 + pSizeHW; const maxX = levelConfig.gameWorldWidth / 2 - pSizeHW;
        const minYw = -levelConfig.WORLD_Y_OFFSET + pSizeHH; const maxYw = levelConfig.gameWorldHeight - levelConfig.WORLD_Y_OFFSET - pSizeHH;
        playerPos.x = clamp(playerPos.x, minX, maxX);
        playerPos.y = clamp(playerPos.y, minYw, maxYw);
        if (Math.abs(dx) > 0.01 || Math.abs(dy) > 0.01) {
            const angle = Math.atan2(dy, dx); let targetRotZ = angle - Math.PI / 2;
            const currentRotZ = playerObject.rotation.z; const twoPi = Math.PI * 2;
            let diff = targetRotZ - currentRotZ;
            while (diff < -Math.PI) diff += twoPi; while (diff > Math.PI) diff -= twoPi;
            playerObject.rotation.z += diff * 0.15;
        }
    }
    playerObject.userData?.mixer?.update(dt);

    let playerCurrentRoom = null;
    if (playerObject?.position && levelData?.rooms) {
        const pX = playerObject.position.x; const pY = playerObject.position.y;
        for (const room of levelData.rooms) {
            const bounds = worldRoomBoundariesRef.current[room.id];
            if (bounds && pX >= bounds.xMinWorld && pX <= bounds.xMaxWorld && pY >= bounds.yMinWorld && pY <= bounds.yMaxWorld) {
                playerCurrentRoom = room.id; break;
```

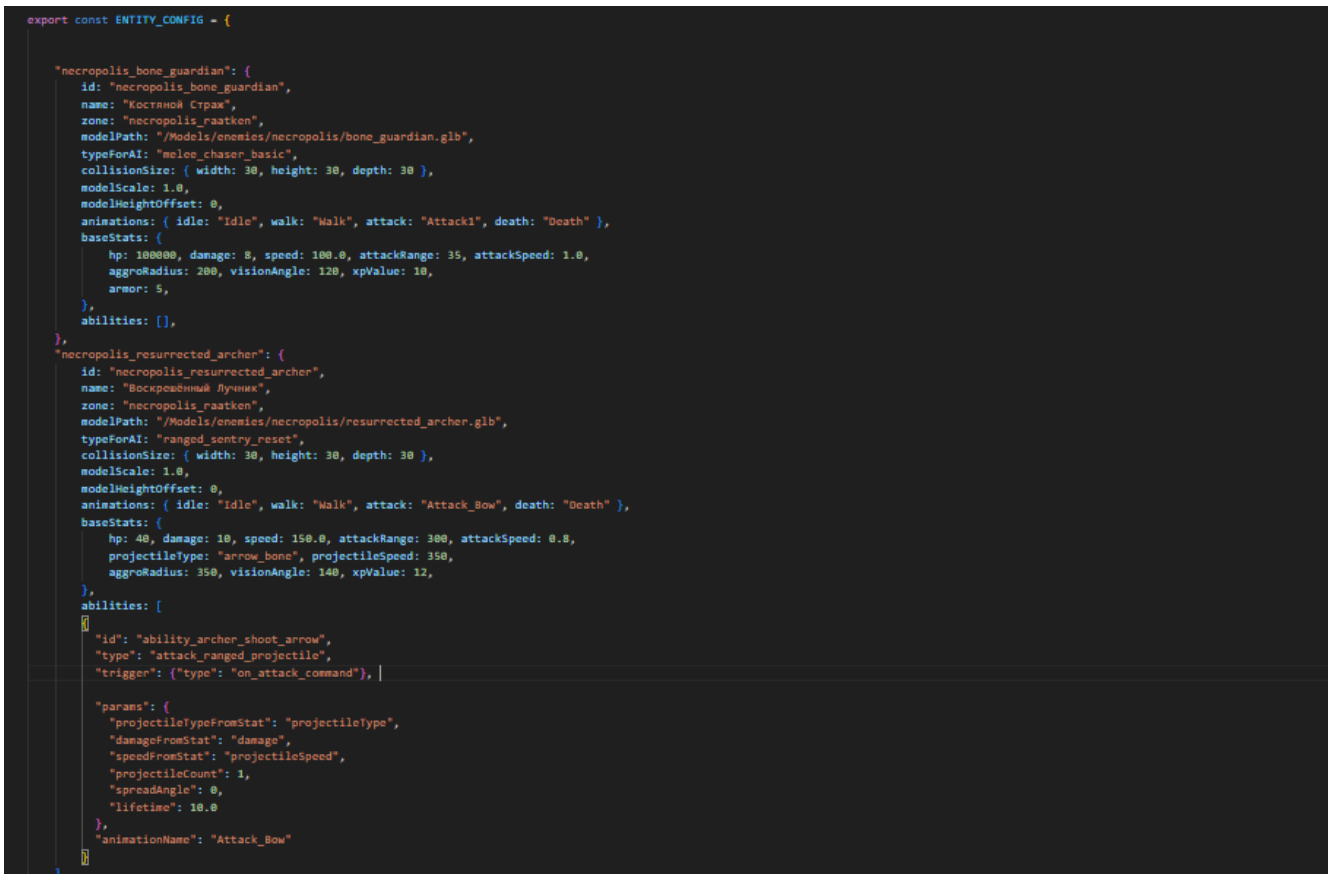
Малюнок 3.4 – Детальніша структура початку функції animate

Джерело: запропоноване автором

```
            });
        }
        const currentAuraStatusInStore = useGameStore.getState().isAffectedByWeakeningAura;
        if (isPlayerCurrentlyInAnyAura !== currentAuraStatusInStore) { setWeakeningAuraStatus(isPlayerCurrentlyInAnyAura); }

        // Door Animations
        const remainingDoorAnimations = [];
        for (const anim of animatingDoorsRef.current) {
            anim.elapsedTime += dt;
            const t = Math.min(anim.elapsedTime / anim.duration, 1.0);
            anim.mesh.position.y = THREE.MathUtils.lerp(anim.startY, anim.targetY, t);
            if (t < 1.0) { remainingDoorAnimations.push(anim); }
            else { anim.mesh.visible = false; }
        }
        if (animatingDoorsRef.current.length !== remainingDoorAnimations.length) { animatingDoorsRef.current = remainingDoorAnimations; }

        // Chest Interaction
        const interactionRadius = 45;
        const interactionRadiusSq = interactionRadius * interactionRadius;
        if (playerObject?.position && levelChestsRef.current?.length > 0) {
            const playerPosChest = playerObject.position;
            levelChestsRef.current.forEach(chest => {
                if (!chest.isOpened && chest.object3D) {
                    const distSq = playerPosChest.distanceToSquared(chest.position);
                    if (distSq <= interactionRadiusSq) {
                        chest.isOpened = true;
                        if (typeof openLevelChest === 'function') { openLevelChest(chest.instanceId, chest.chestTypeId); }
                        if(chest.object3D && chest.object3D.material) {
                            chest.object3D.material.transparent = true;
                            chest.object3D.material.opacity = 0.5;
                        }
                    }
                }
            });
        }

        // 6. Check Win/Loss
        checkWinCondition();

        // 7. Update Camera
        if (playerObject && currentCamera && levelConfig) {
            const camWidth = currentCamera.right - currentCamera.left;
            const camHeight = currentCamera.top - currentCamera.bottom;
            const targetXCam = clamp(playerPos.x, -levelConfig.gameWorldWidth / 2 + camWidth / 2, levelConfig.gameWorldWidth / 2 - camWidth / 2);
            const targetYCam = clamp(playerPos.y, -levelConfig.WORLD_Y_OFFSET + camHeight / 2, levelConfig.gameWorldHeight - levelConfig.WORLD_Y_OFFSET - camHeight / 2);
            currentCamera.position.lerp(new THREE.Vector3(targetXCam, targetYCam, currentCamera.position.z), 0.1);
        }

        // 8. Render
        if (currentRenderer && currentScene && currentCamera) {
            try { currentRenderer.render(currentScene, currentCamera); }
            catch (error) {
                console.error("X Ошибка рендеринга:", error); setLevelStatus('error');
                if(animationFrameId.current) cancelAnimationFrame(animationFrameId.current);
                animationFrameId.current = null; clock.stop();
            }
        }
    };

    if (!animationFrameId.current) {
        clock.start();
```

Малюнок 3.5 – Детальніша структура функції animate

Джерело: запропоноване автором



```
export const ENTITY_CONFIG = {


    "necropolis_bone_guardian": {
        id: "necropolis_bone_guardian",
        name: "Костяной Страж",
        zone: "necropolis_raatken",
        modelPath: "/Models/enemies/necropolis/bone_guardian.glb",
        typeForAI: "melee_chaser_basic",
        collisionSize: { width: 30, height: 30, depth: 30 },
        modelScale: 1.0,
        modelHeightOffset: 0,
        animations: { idle: "Idle", walk: "Walk", attack: "Attack1", death: "Death" },
        baseStats: {
            hp: 100000, damage: 8, speed: 100.0, attackRange: 35, attackSpeed: 1.0,
            aggroRadius: 200, visionAngle: 120, xpValue: 10,
            armor: 5,
        },
        abilities: [],
    },
    "necropolis_resurrected_archer": {
        id: "necropolis_resurrected_archer",
        name: "Воскрешённый Лучник",
        zone: "necropolis_raatken",
        modelPath: "/Models/enemies/necropolis/resurrected_archer.glb",
        typeForAI: "ranged_sentry_reset",
        collisionSize: { width: 30, height: 30, depth: 30 },
        modelScale: 1.0,
        modelHeightOffset: 0,
        animations: { idle: "Idle", walk: "Walk", attack: "Attack_Bow", death: "Death" },
        baseStats: {
            hp: 40, damage: 10, speed: 150.0, attackRange: 300, attackSpeed: 0.8,
            projectileType: "arrow_bone", projectileSpeed: 350,
            aggroRadius: 350, visionAngle: 140, xpValue: 12,
        },
        abilities: [
        {
          "id": "ability_archer_shoot_arrow",
          "type": "attack_ranged_projectile",
          "trigger": {"type": "on_attack_command"},

          "params": {
            "projectileTypeFromStat": "projectileType",
            "damageFromStat": "damage",
            "speedFromStat": "projectileSpeed",
            "projectileCount": 1,
            "spreadAngle": 0,
            "lifetime": 10.0
          },
          "animationName": "Attack_Bow"
        }
```

Малюнок 3.6 Структура ENTITY_CONFIG

Джерело: запропоноване автором

Хук useEffect ідеально підходить для запуску та зупинки ігрового циклу. Функція animate, викликана через requestAnimationFrame, виконується перед кожним перемалюванням екрану браузером (зазвичай 60 разів на секунду). Всередині цієї функції послідовно викликаються функції, що відповідають за різні підсистеми гри: оновлення гравця, ворогів (де і викликається логіка з AI_PATTERNS), снарядів, перевірку зіткнень та, нарешті, рендеринг фінального зображення. Такий структурований підхід дозволяє чітко контролювати порядок виконання операцій у кожному кадрі, що є ключовим для створення стабільного та передбачуваного ігрового процесу.

3.3 Використання програмного додатку

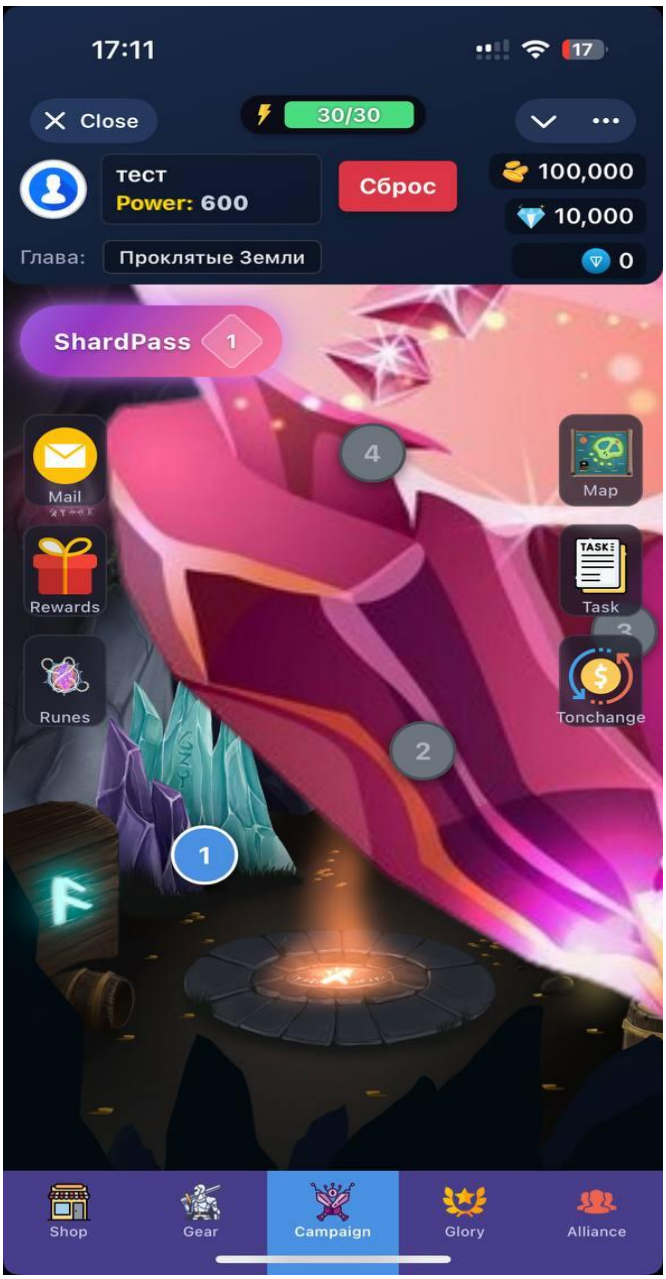
Після завершення розробки та тестування ключового функціоналу, гейміфікована платформа готова до використання кінцевим користувачем — гравцем. Цей підрозділ демонструє практичне застосування розробленого програмного продукту, ілюструючи основні сценарії взаємодії гравця з ігровим світом та P2E-механіками.

Фінальний візуальний стиль, моделі персонажів та ворогів знаходяться на етапі активної розробки. Зокрема, замість фінальних моделей ворогів на даному етапі використовуються прості геометричні меші, що дозволяє зосередитись на відточуванні та тестуванні основних ігрових та бойових механік.

Головний екран та навігація

Після запуску додатку та автоматичної авторизації гравець потрапляє на головний екран кампанії (мал. 3.7). Інтерфейс надає доступ до всієї ключової інформації та функцій:

- У верхній частині: відображається інформація про профіль гравця (ім'я "тест", показник сили Power: 600), а також баланси основних ігрових валют.
- У центрі: знаходиться основний навігаційний елемент для вибору рівнів або глав кампанії.
- З боків та внизу: розташовані кнопки для швидкого доступу до основних розділів: Shop (Магазин), Gear (Екіпірування), Glory (Досягнення), Mail (Пошта), Task (Завдання) та ключової функції Tonchange (Обмін).

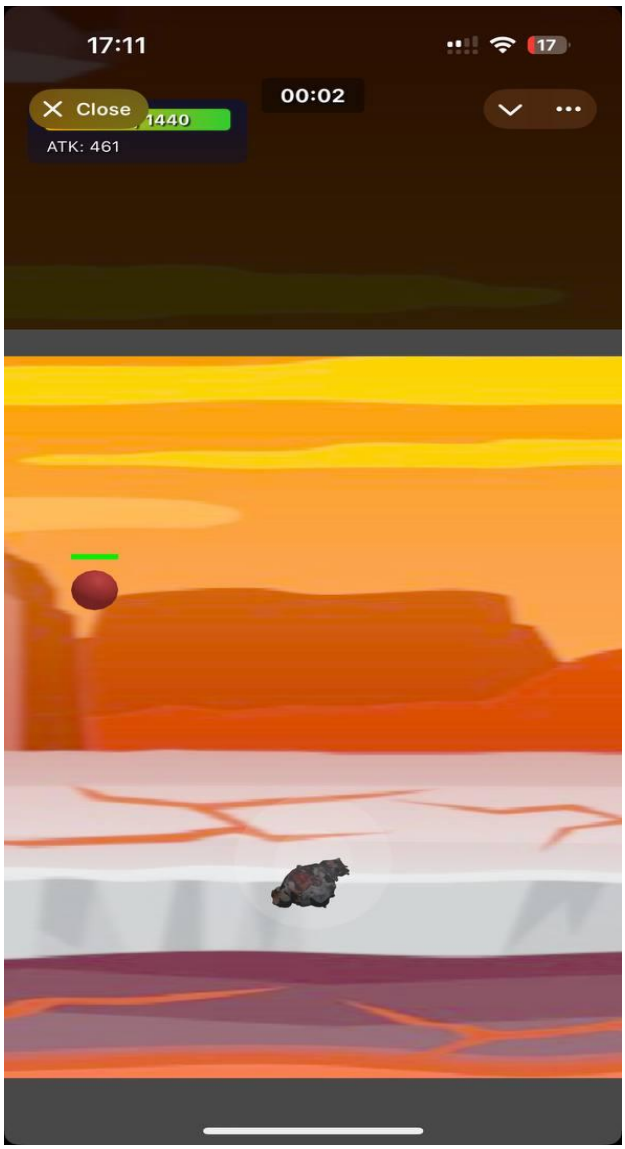


Малюнок 3.7 – Головний екран кампанії

Джерело: запропоноване автором

Основний ігровий процес

Запуск рівня переносить гравця на ігровий екран (мал. 3.8). Тут відбувається основна дія: гравець керує своїм персонажем, переміщаючись по локації та атакуючи ворогів. На скріншоті видно персонажа гравця (представленого тимчасовою моделлю), який атакує ворога (представленого у вигляді простого меша-кулі для тестування механік). У верхній частині екрану відображається здоров'я гравця та таймер рівня.



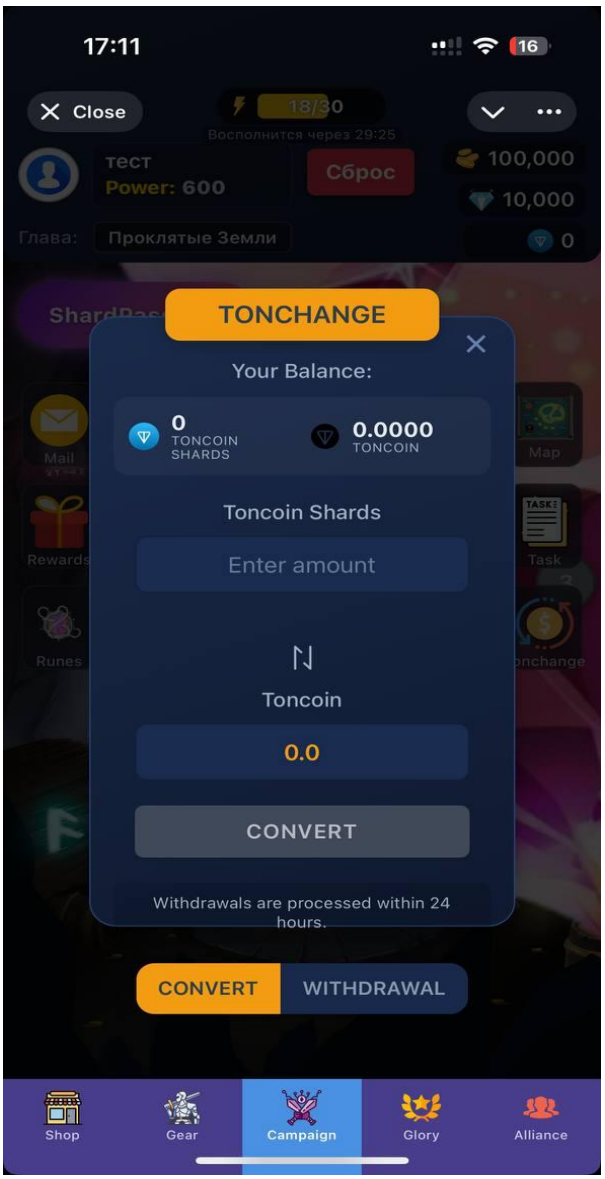
Малюнок 3.8 – Приклад ігрового процесу

Джерело: запропоноване автором

Реалізація P2E-механік: обмін та виведення

Ключовою особливістю платформи є можливість монетизації ігрового часу. Цей процес реалізовано у два етапи через інтерфейс Tonchange.

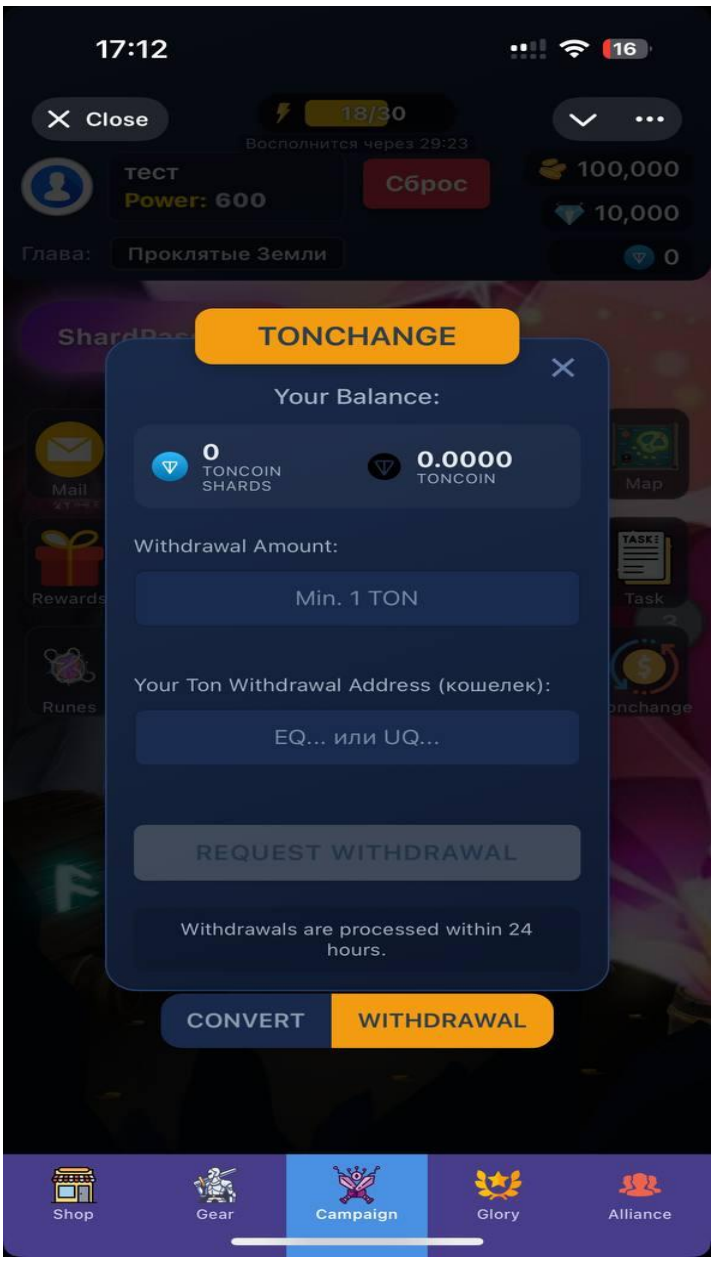
Етап 1: Внутрішній обмін. Спочатку гравець обмінює зароблені в грі ресурси TONCOIN SHARDS на внутрішній баланс TONCOIN (мал. 3.9). Це off-chain операція, що відбувається миттєво всередині системи.



Малюнок 3.9 – Інтерфейс внутрішнього обміну ресурсів (вкладка Convert)

Джерело: запропоноване автором

Етап 2: Виведення на гаманець. Після накопичення коштів на внутрішньому балансі, гравець може перейти на вкладку WITHDRAWAL та створити запит на виведення TONCOIN на свій реальний TON-гаманець (рис. 3.10), вказавши суму та адресу гаманця. Це on-chain операція, що обробляється протягом визначеного часу.



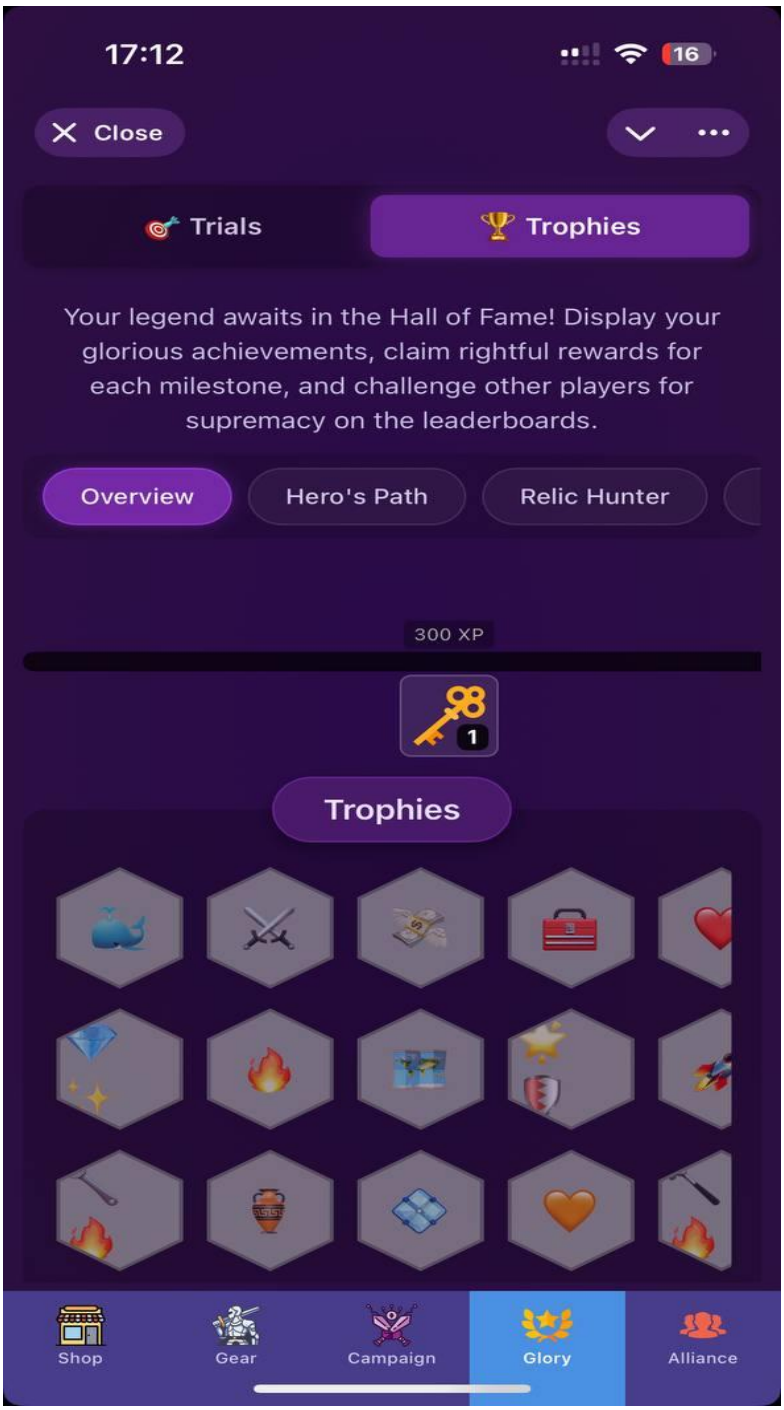
Малюнок 3.10 – Інтерфейс виведення коштів на гаманець (вкладка Withdrawal)

Джерело: запропоноване автором

Додатковий функціонал платформи

Окрім основного ігрового циклу, платформа містить низку додаткових систем, що збагачують ігровий досвід:

- Система досягнень (Glory): (мал. 3.11) відстежує успіхи гравця, нагороджуючи його за виконання певних етапів та завдань.



Малюнок 3.11 – Система досягнень та трофеїв

Джерело: запропоноване автором

Таким чином, представлені скріншоти демонструють, що розроблена гейміфікована платформа є функціональним та логічно завершеним продуктом, що реалізує всі ключові сценарії, від ігрового процесу до P2E-механік.

ВИСНОВКИ

У даній кваліфікаційній роботі було проведено комплексне дослідження, проектування та розробку прототипу гейміфікованої платформи на базі Telegram з використанням технології блокчейн. Метою роботи було створення програмного продукту, що поєднує захоплюючий ігровий процес жанру Action RPG з доступністю платформи Telegram Mini Apps та економічними стимулами моделі Play-to-Earn. За результатами виконаної роботи можна зробити наступні висновки.

1. Проведено комплексний аналіз предметної області. У першому розділі було досліджено ринок GameFi, особливості платформи Telegram Mini Apps та архітектуру існуючих програмних аналогів. Аналіз підтвердив високу актуальність проекту та виявив вільну ринкову нішу для продукту, що поєднує глибокий ігровий процес зі зручністю та віральністю месенджера.
2. Спроектовано архітектуру та моделі системи. У другому розділі було розроблено повний набір проектних рішень. За допомогою мови UML були створені діаграми варіантів використання та діяльності, що формалізували функціональність та основні бізнес-процеси. Була запропонована та

обґрунтована трирівнева клієнт-серверна архітектура системи. Також було розроблено детальну ER-діаграму та спроектовано структуру бази даних PostgreSQL, що є фундаментом для зберігання всіх ігрових даних.

3. Здійснено програмну реалізацію ключових компонентів. У третьому розділі було описано практичну реалізацію проекту. Була продемонстрована архітектура програмного коду, заснована на принципах модульності та розділення відповідальності. Наведено приклади програмної реалізації (лістинги коду) для серверної частини, що концептуально описує логіку API на Python/FastAPI, та для клієнтської частини, що демонструє реалізацію ігрового циклу та штучного інтелекту на JavaScript/React/Three.js. Також було продемонстровано роботу готового прототипу за допомогою скріншотів інтерфейсу.

Наукова новизна роботи полягає у запропонованому підході до інтеграції складного ігрового процесу жанру Action RPG, що вимагає рендерингу 3D-графіки та обробки логіки в реальному часі, в обмежене середовище платформи Telegram Mini Apps. Розроблена двохетапна модель проведення P2E-транзакцій (off-chain обмін та on-chain виведення) є ефективним архітектурним рішенням, що оптимізує витрати та покращує користувацький досвід.

Практичне значення отриманих результатів полягає у створенні готового до подальшого тестування та запуску прототипу програмного продукту з високим комерційним потенціалом. Крім того, розроблені архітектурні рішення, моделі даних та програмні компоненти (зокрема, система AI-патернів) можуть слугувати методичною та практичною основою для інших розробників, що прагнуть створювати складні ігрові проекти для екосистеми Telegram.

Варто зазначити певні обмеження проведеної роботи. Представлена реалізація є прототипом (MVP), який демонструє основний функціонал, але потребує подальшого розширення контентом. Серверна частина була описана на концептуальному рівні та потребує повної імплементації та розгортання.

Економічна модель P2E є теоретичною та потребує подальшого тестування та балансування в умовах реального ринку та поведінки гравців.

В якості напрямів для подальшого розвитку проекту можна виділити наступне:

— Повноцінна реалізація та розгортання серверної частини на хмарній інфраструктурі.

— Розширення ігрового контенту: додавання нових рівнів, ворогів, предметів та квестів.

— Впровадження більш складних блокчейн-механік, таких як NFT для унікальних ігрових предметів.

— Розробка багатокористувацьких режимів (PvP-арени, кооперативні рейди).

— Впровадження комплексної системи автоматизованого тестування.

Таким чином, дана кваліфікаційна робота повністю досягла поставленої мети, продемонструвавши повний цикл розробки складного програмного продукту від ідеї та всебічного аналізу до детального проектування та практичної реалізації функціонального прототипу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram. Telegram Bot API Documentation – url: https://core.telegram.org/bots/api
2. Telegram. Telegram Web Apps API Documentation – url: https://core.telegram.org/bots/webapps
3. The Open Network (TON). TON Blockchain Documentation – url: https://ton.org/docs/
4. The Open Network (TON). FunC Documentation (for TON Smart Contracts) – url: https://ton.org/docs/develop/func/
5. The Open Network (TON). Tact Documentation (for TON Smart Contracts) – url: https://ton.org/docs/develop/tact/
6. TonWeb. JavaScript SDK for The Open Network (TON) – url: https://github.com/toncenter/tonweb
7. PixiJS. Офіційна документація – url: https://pixijs.com/ (або https://pixijs.download/release/docs/index.html для API)
8. Three.js. Офіційна документація – url: https://threejs.org/docs/
9. Python Software Foundation. Python Language Reference – url: https://docs.python.org/3/reference/index.html
10. FastAPI. FastAPI Documentation – url: https://fastapi.tiangolo.com/
11. Django Project. Django Documentation – url: https://docs.djangoproject.com/en/stable/
12. MongoDB. MongoDB Manual – url: https://www.mongodb.com/docs/manual/
13. PostgreSQL. PostgreSQL Documentation – url: https://www.postgresql.org/docs/
14. Solidity. Solidity Documentation – url: https://docs.soliditylang.org/
15. Web3.js. Web3.js Documentation – url: https://web3js.readthedocs.io/
16. BNB Chain. BNB Chain Documentation – url: https://docs.bnbchain.org/
17. Polygon. Polygon Technology Documentation – url: https://wiki.polygon.technology/
18. IPFS. IPFS Documentation [Електронний ресурс]. – url: https://docs.ipfs.tech/

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Назва програмного продукту: "Гейміфікована платформа на базі Telegram з використанням технології блокчейн".

Замовник: Сумський національний аграрний університет, кафедра кібернетики та інформатики.

Виконавець: студент групи ІСТ 2101 с. т. Черкашин Іван Олександрович.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є виконання кваліфікаційної роботи освітнього ступеня «Бакалавр» за спеціальністю 126 «Інформаційні системи та технології».

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Програмний продукт розробляється з метою створення інноваційної ігрової платформи, що поєднує глибокий ігровий процес жанру Action RPG з масовою доступністю месенджера Telegram та економічними можливостями моделі Play-to-Earn (P2E).

Система призначена для широкого кола користувачів месенджера Telegram, зацікавлених в іграх та можливостях заробітку на основі технології блокчейн. Платформа має на меті вирішити проблему високого порогу входу в існуючі Web3-ігри, надаючи зручний та інтуїтивно зрозумілий інтерфейс.

4. ВИМОГИ ДО СИСТЕМИ

4.1. Функціональні вимоги

4.1.1. Вимоги до ігрового процесу

Система повинна реалізовувати ігровий процес у жанрі Action RPG.

Користувач (Гравець) повинен мати можливість керувати своїм персонажем у реальному часі.

Має бути реалізована бойова система з атаками та використанням навичок.

Система повинна містити механіку прогресії персонажа, включаючи підвищення показника сили (power).

Має бути реалізована система квестів та отримання ігрових предметів.

4.1.2. Вимоги до P2E-економіки

В системі має бути реалізований внутрішньоігровий ресурс Shards, що заробляється гравцями.

Має бути реалізований механізм внутрішнього (off-chain) обміну Shards на внутрішній баланс TON.

Має бути реалізований механізм виведення (on-chain) коштів з внутрішнього балансу TON на реальний гаманець користувача в мережі The Open Network.

Система повинна забезпечувати безпечну інтеграцію з TON-гаманцями.

4.1.3. Вимоги до інтерфейсу користувача

Система повинна забезпечувати безшовну авторизацію користувачів через їхній обліковий запис Telegram.

Інтерфейс має бути адаптивним та коректно відображатися на мобільних пристроях.

Платформа повинна містити наступні екрани: головне меню, екран персонажа та екіпірування, ігровий екран, інтерфейс обміну та виведення коштів.

4.1.4. Вимоги до адміністративної панелі

Адміністратор повинен мати доступ до захищеної веб-панелі.

Панель повинна надавати функціонал для управління ігровим контентом (предмети, квести).

Панель повинна надавати можливість перегляду статистики по гравцях та моніторингу ключових економічних показників.

4.2. Нефункціональні вимоги

Продуктивність: Ігровий клієнт має забезпечувати стабільну частоту кадрів (не нижче 30 FPS) на сучасних мобільних пристроях. Час відгуку API на запити клієнта не повинен перевищувати 300 мс.

Масштабованість: Архітектура серверної частини повинна бути розрахована на підтримку від 1000 одночасних користувачів з можливістю подальшого масштабування.

Надійність: Система має забезпечувати доступність на рівні 99.5%. Мають бути передбачені механізми резервного копіювання бази даних.

4.3. Вимоги до безпеки

Взаємодія між клієнтом та сервером має відбуватися за захищеним протоколом HTTPS.

Система має бути захищена від поширених веб-вразливостей (XSS, CSRF, SQL Injection).

Ключі доступу, паролі та інші чутливі дані мають зберігатися у зашифрованому вигляді.

Взаємодія з блокчейном має відбуватися через безпечні серверні механізми, виключаючи зберігання приватних ключів користувачів.

5. ВИМОГИ ДО ТЕХНОЛОГІЙ

Серверна частина: Python 3.9+, FastAPI, Django (для адмін-панелі), SQLAlchemy.

Клієнтська частина: JavaScript (ES6+), React, Three.js, Pixi.js.

База даних: PostgreSQL 14+.

Блокчейн: The Open Network (TON).

Середовище розробки: Visual Studio Code, Git.

6. ЕТАПИ ТА ТЕРМІНИ РОЗРОБКИ

Розробка програмного продукту проводиться відповідно до етапів, визначених у календарному плані. Детальний календарний план робіт представлено у Додатку Б "Планування робіт".

7. ПОРЯДОК ПРИЙМАННЯ ТА ТЕСТУВАННЯ

Тестування включає модульне, інтеграційне та функціональне тестування ключових компонентів.

Працездатність системи підтверджується демонстрацією виконання основних користувацьких сценаріїв, описаних у розділі 3.3 кваліфікаційної роботи.

Робота вважається прийнятою після успішного публічного захисту кваліфікаційної роботи перед екзаменаційною комісією.

ДОДАТОК Б

ПЛАНУВАННЯ РОБІТ

Управління проектом є невід'ємною частиною успішної розробки складних програмних систем. Для візуалізації плану робіт, розподілу завдань у часі та відстеження прогресу в рамках даного проекту було використано діаграму Ганта. Цей інструмент дозволяє наочно представити послідовність виконання завдань, їхню тривалість та взаємозв'язки.

На малюнку Б.1 представлено календарний план-графік виконання ключових етапів розробки функціоналу гейміфікованої платформи.

Малюнок Б.1 – Діаграма Ганта виконання робіт з розробки проекту

Джерело: запропоноване автором

Представлена діаграма Ганта деталізує послідовність розробки основних ігрових механік та систем, пов'язаних з персонажем та внутрішньоігровою економікою. План робіт можна логічно розділити на декілька ключових фаз:

Фаза 1: Розробка базового функціоналу персонажа (Q1 2025). Цей етап є фундаментальним і включає реалізацію всіх систем, пов'язаних з гравцем:

- Створення системи екіпірування (Character equipment, Gear).
- Розробка інтерфейсу для перегляду персонажа та його характеристик (Character viewer, Stats, Artifacts).
- Реалізація взаємодії з екіпіруванням через відповідні елементи управління (Gear button and scroll with gear).

Фаза 2: Реалізація системи економіки та монетизації (Q1-Q2 2025). На цьому етапі розробляються механізми, що формують внутрішньоігрову економіку:

- Створення систем для продажу наборів (Bundles) та спеціальних пропозицій (One-time bundles, Daily offers).
- Реалізація скринь з нагородами (Gear chest).
- Впровадження основних ігрових валют, включаючи Gold, Diamonds та ключовий P2E-ресурс Tonshards.

Фаза 3: Розробка основного ігрового циклу (Q2 2025). Це заключний етап, де всі розроблені системи інтегруються в єдиний ігровий процес:

- Створення головного компонента ігрового рівня (Main level component).
- Розробка серверної логіки для підтримки цього компонента.

ДОДАТОК В

КОПІЇ ПУБЛІКАЦІЙ

Міністерство внутрішніх справ України
Харківський національний університет внутрішніх справ
Сумська філія
Сумський осередок Європейської асоціації студентів права

ПИТАННЯ СУЧАСНОЇ НАУКИ І ПРАВА

Матеріали
XV Всеукраїнської науково-практичної конференції
здобувачів вищої освіти

25 квітня 2025 року

Суми 2025

Іван Олександрович Черкашин – *здобувач вищої освіти IV курсу Сумського національного аграрного університету*
Науковий керівник: **Олександр Борисович В'юненко** – *доцент кафедри кібернетики та інформатики Сумського національного аграрного університету, кандидат економічних наук, доцент*

TELEGRAM MINI APPS ЯК ПЛАТФОРМА ДЛЯ ІНТЕРАКТИВНИХ ФІНАНСОВИХ СЕРВІСІВ НА БАЗІ БЛОКЧЕЙНУ

Telegram уже давно перестав бути просто месенджером. Завдяки відкритому API, ботам та Web Apps, платформа перетворилася на повноцінне середовище для розробки інтерактивних сервісів. Особливої популярності останнім часом набули Telegram Mini Apps — вебзастосунки, які інтегруються безпосередньо у месенджер і працюють як частина його інтерфейсу.

Mini Apps відкривають нові можливості не лише для гейміфікації чи комунікації, а й для впровадження фінансових сервісів, які можуть бути інтегровані з блокчейн-технологіями. У контексті розвитку Web3 Telegram виступає перспективним хабом для надання децентралізованих фінансових послуг (DeFi) через звичний для користувача інтерфейс.

Telegram Mini Apps працюють через WebView, а це дозволяє створювати вебінтерфейси з будь-якою логікою — від гаманців і токенізованих активів до бірж, кредитних сервісів, краудфандингу чи систем мікроплатежів. Інтеграція таких додатків із блокчейном відбувається за допомогою бібліотек Web3.js або Ethers.js, які дозволяють зчитувати стан смартконтрактів, проводити транзакції, перевіряти підпис користувача.

Особливе значення має блокчейн TON (The Open Network), який розвивається як рідна екосистема для Telegram. На відміну від Ethereum чи BSC, він має високу швидкість, низькі комісії та глибоку інтеграцію з Telegram ID, що робить його зручним для фінансових застосунків прямо в месенджері.

Переваги використання Telegram Mini Apps для фінансових сервісів.

Однією з головних переваг використання Telegram Mini Apps у фінансових рішеннях є максимальна доступність для кінцевого користувача. Завдяки тому, що Telegram встановлений на мільйонах пристроїв по всьому світу, немає потреби в окремих інсталяціях застосунків чи реєстраціях – достатньо лише натиснути кнопку запуску в боті або перейти за посиланням. Це спрощує вхід у фінансову систему для багатьох користувачів, особливо тих, хто має низький технічний рівень або обмежений доступ до класичних банківських інструментів.

З точки зору інтерфейсного дизайну (UI/UX), Mini Apps підтримують розробку повноцінного мобільного інтерфейсу, побудованого на HTML5, CSS і JavaScript. Це дозволяє використовувати популярні фреймворки, такі

як React, Vue або навіть Pixi.js для анімованих інтерфейсів. Розробники отримують повну свободу у створенні зручних, інтуїтивно зрозумілих і адаптивних екранів, що підвищує довіру до фінансового сервісу.

Інтеграція з ботами Telegram забезпечує багатофункціональну підтримку користувача. Наприклад, бот може надсилати push-сповіщення про нові транзакції, оновлення балансу, курси валют або нагадування про необхідність підписати смартконтракт. Така автоматизована взаємодія значно підвищує зручність і функціональність платформи.

Ще одним ключовим фактором є підтримка монетизації: Telegram дозволяє приймати платежі через Payments API, а також — за умови впровадження блокчейн-рішень — підтримувати криптогаманці та токени. Це робить платформу перспективною не лише для користувача, а й для розробників і бізнесу.

Приклади фінансових сервісів, які можливо реалізувати у Telegram Mini Apps.

Telegram Mini Apps можуть стати середовищем для створення широкого спектра фінансових рішень: Некастодіальні криптогаманці — дозволяють користувачам зберігати приватні ключі без передачі контролю третім особам. P2P-обмінники — сервіси, що забезпечують прямий обмін криптовалют між користувачами за вигідним курсом. DeFi-продукти — як-от стейкінг, ліквідність-пули, фармінг токенів та управління NFT-активами. DAO або інвестиційні клуби — де користувачі можуть колективно голосувати за розподіл коштів або обирати стратегії. Краудфандинг-платформи — проєкти збору коштів на основі смартконтрактів із прозорою аналітикою та NFT-нагородами.

Список використаних джерел:

1. Telegram. Web Apps API. Documentation : веб-сайт. URL: https://core.telegram.org/bots/webapps (дата звернення: 17.03.2025).

2. TON. The Open Network Docs. URL: https://docs.ton.org/ (дата звернення: 18.03.2025).

3. Web3.js – API JavaScript Ethereum. Documentation : веб-сайт. URL: https://web3js.readthedocs.io/ (дата звернення: 17.03.2025).

4. Ethers.js. Documentation : веб-сайт. URL: https://docs.ethers.org/ (дата звернення: 17.03.2025).

5. Solidity. Official Documentation : веб-сайт. URL: https://soliditylang.org/docs/ (дата звернення: 20.03.2025).

6. Firebase. Firebase Documentation : веб-сайт. URL: https://firebase.google.com/docs (дата звернення: 18.03.2025).

7. Telegram Bot Payments API. : веб-сайт. URL: https://core.telegram.org/bots/payments (дата звернення: 20.03.2025).

SCIENCE AND TECHNOLOGY: CHALLENGES, PROSPECTS AND INNOVATIONS

Proceedings of XI International Scientific and Practical Conference

Osaka, Japan

19-21 June 2025

Osaka, Japan

2025

УДК 519.876:004.94:005.8

AN ARCHITECTURAL ANALYSIS OF THE TELEGRAM MINI APPS PLATFORM AND ITS SYNERGY WITH THE OPEN NETWORK (TON) FOR DEVELOPING SECURE, SCALABLE, AND INTERACTIVE FINANCIAL SERVICES WITH A LOW BARRIER TO ENTRY

Cherkashyn Ivan Olexandrovich
Student
Sumy National Agrarian University
Sumy, Ukraine
Viunenko Oleksandr Borisovich
Associate Professor, Department of Cybernetics and Informatics
Sumy National Agrarian University

Despite its long-standing potential, the mass adoption of blockchain technology and decentralized finance (DeFi) has been limited by high barriers to entry for mainstream users, such as the complexity of managing crypto wallets, the need to understand transaction fees (gas), and a fragmented user experience. In this context, a new paradigm is emerging: the synergy between the Telegram Mini Apps (TMA) platform and The Open Network (TON) blockchain, which presents a unique solution to this persistent problem. This architectural fusion is not just another technical solution, but a fundamental paradigm shift in the development and distribution of decentralized applications. This model creates a unique environment for developing secure, scalable, and truly interactive financial services that, by drastically lowering the barrier to entry, are capable of attracting hundreds of millions of new users to the Web3 economy and permanently changing the landscape of digital finance.

The era of Web3 and DeFi arrived with grand promises: to create a more open, fair, and transparent financial system, free from the control of intermediaries. The idea that anyone with an internet connection could become their own bank, control their own assets, and participate in the global economy on equal footing was, and remains, revolutionary. However, a deep chasm has formed between this vision and reality. For the vast majority of the global population, the world of cryptocurrency has remained terra incognita—a territory

that is complex, risky, and frankly hostile to newcomers. The primary obstacle has been the overwhelming complexity of the user experience (UX). The process of entering the DeFi world was akin to navigating an obstacle course. First, there was the management of crypto wallets. A user was required to install a browser extension or a mobile app, and then confront the most responsible and intimidating part: saving a "seed phrase" of 12 or 24 random words. Losing this phrase meant the irreversible loss of all funds, while its compromise meant certain theft. This level of personal responsibility, unfamiliar to users of traditional web services with their "Forgot Password?" function, became an insurmountable psychological barrier.

Second, there was the economy of transactions. The concept of "gas"—a fee for executing operations on the network, the cost of which constantly fluctuates depending on blockchain congestion—baffled newcomers. Why could a simple token transfer cost $5 today and $50 tomorrow? What is Gwei, slippage, and how does one choose the correct gas limit? These questions turned basic financial operations into a stressful process requiring specialized knowledge. Finally, the fragmentation of the ecosystem. To complete a single task, such as swapping one token for another and investing it in a liquidity pool, a user often had to interact with multiple sites (dApps), each with its own interface, connect their wallet to them, and sign numerous incomprehensible transactions. The absence of a single, seamless pathway led to confusion and created a fertile ground for phishing and scams. All these factors combined to create an impenetrable wall that effectively cut off the mass audience from the innovations of Web3.

It is precisely in this context that the "mass adoption" problem found its most elegant solution to date. It was born not from the creation of another "Ethereum killer" or a new complex protocol, but from the fusion of two seemingly independent giants: the social platform Telegram and the resilient blockchain, The Open Network (TON). Telegram has long since evolved beyond being just a messenger. With an audience exceeding 900 million active users, it has transformed into a full-fledged ecosystem with channels, bots, and, most importantly, the Mini Apps platform. TMAs are essentially web applications that run directly within the Telegram interface. They do not require downloading, installation, or separate registration. A user simply clicks a link and is instantly inside a fully functional application.

For Web3, this is a game-changer, eliminating the primary barrier of initial friction.

If TMA is the perfect delivery channel, then TON is the engine and infrastructure capable of servicing that channel. Originally designed for integration with Telegram and later handed over to an independent developer community, this blockchain was engineered from the ground up to handle mass workloads. Its key architectural features, like dynamic sharding, allow it to process millions of transactions per second, a necessary condition for serving Telegram's vast user base. Furthermore, TON's architecture is designed for consistently low transaction fees, making microtransactions economically viable. The ecosystem completes this user-friendly approach with two key components: TON Space, a self-custodial wallet integrated directly into Telegram's settings, and TON Connect, a protocol that makes connecting the wallet to an app as simple and secure as using Apple Pay. Together, Telegram provides a seamless interface and a colossal user base, while TON provides a scalable, fast, and cheap infrastructure, effectively eliminating the key barriers that have hindered Web3 adoption.

The theoretical elegance of this union, however, would be meaningless without practical proof of its effectiveness. This proof came in the form of a phenomenon that shook the crypto industry in early 2024: the game Notcoin. At its core, Notcoin was a brilliantly simple "clicker" launched as a Mini App in Telegram. Users tapped on an animated coin to earn in-game points, with simple boosts and a viral referral program. The result exceeded all expectations, attracting over 35 million users in a few months. People who had never heard of crypto were enthusiastically "mining" virtual coins. Notcoin became the perfect demonstration of the TMA and TON synergy. It spread virally through Telegram's social graph, and it brilliantly gamified the complex concept of an airdrop; users weren't "claiming an airdrop," they were "playing a game." This led to the seamless onboarding of millions into Web3, as they activated their TON Space wallets to claim their eventual rewards, becoming owners of a real crypto asset overnight. The success of Notcoin spawned a wave of "Tap-to-Earn" projects like Hamster Kombat, which gamified even more complex DeFi concepts like yield farming, masking them as simple game upgrades. This approach, dubbed SocialFi (Social Finance), uses social interactions and game mechanics to engage users in financial products, marking a fundamental departure from traditional, expert-oriented DeFi.