КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:

Telegram-бот для гейміфікованої взаємодії з користувачами: концепція, архітектура та реалізація

КОЗАР Олександр Андрійович

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., професор Гончаренко Т.А.

„___” _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: **«Telegram-бот для гейміфікованої взаємодії з користувачами: концепція, архітектура та реалізація»**

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач
Козар Олександр Андрійович
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21-2
Керівник Рябчун Ю.В.
(прізвище та ініціали)
Доктор філософії
(вчене звання, науковий ступінь)
Рецензент к.т.н., доц. Доля О.В.
(Прізвище та ініціали)

Ідентичність підтверджую

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет:	Автоматизації інформаційних технологій
Випускова кафедра:	Інформаційних технологій
Освітній ступінь:	Бакалавр
Спеціальність:	Комп’ютерні науки
Освітня програма:	Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО

_____

„___”_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА

ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

	Козара Олександра Андрійовича
1. Telegram-бот для гейміфікованої взаємодії з користувачами: концепція, архітектура та реалізація ія	

затверджена наказом ректора КНУБА № 235/23/25 від «14» лютого 2025 року

2. Керівник роботи	Рябчун Юлія Володимирівна, PhD

3. Строк подання Здобувачем роботи до захисту *травень 2025 р.*

4. Зміст пояснювальної записки за розділами:

Р.1	АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ
Р.2	ПРОЕКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Р.3	РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Р.4	ЕРГОНОМІКА ІТ ТА ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

5. Графічний матеріал за розділами:

Р.1. 3 рисунків

Р.2. 9 рисунків, 1 таблиць

Р.3. 6 рисунків

Р.4. 8 таблиці

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	Лютий 2025
Розділ 2	Березень 2025
Розділ 3	Травень 2025
Розділ 4	Травень 2025
Остаточне оформлення роботи	Травень 2025
Направлення роботи для перевірки на плагіат	28.05.2025
Попередній захист роботи на випусковій кафедрі	06.06.2025
Направлення роботи на рецензування	06.06.2025

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Рябчун Ю.В., доц. каф.ІТ	05.02.25	
Розділ 2	Рябчун Ю.В., доц. каф.ІТ	20.03.25	
Розділ 3	Рябчун Ю.В., доц. каф.ІТ	16.05.25	
Розділ 4	Рябчун Ю.В., доц. каф.ІТ	26.05.25	

8. Дата видачі завдання листопад 2024 р.

Зав. кафедри			Гончаренко Т.А.
	(підпис)		(прізвище та ініціали)
Керівники			Рябчун Ю.В.
	(підпис)		(прізвище та ініціали)
Здобувач			Козар О.А.
	(підпис)		(прізвище та ініціали)

АНОТАЦІЯ

Козар О.А. Telegram-бот для гейміфікованої взаємодії з користувачами: концепція, архітектура та реалізація.

Кваліфікаційна випускна робота бакалавра за спеціальністю 122 «Комп’ютерні науки», освітня програма «Інформаційні управляючі системи та технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Дана дипломна робота присвячена розробці інтерактивного Telegram-бота «Happy Dog Bot», який об’єднує клікер-гру на платформі Unity з месенджером Telegram задля підвищення залученості користувачів та створення новаторських каналів монетизації через систему новин й рейтингові списки найактивніших гравців.

Технічне рішення реалізовано за клієнт–серверною архітектурою: Unity (C#) використовує корутини для асинхронних HTTP-запитів до REST API на Java/Spring Boot з авторизацією й логуванням, PostgreSQL (Supabase) забезпечує надійну обробку й збереження даних, а Docker з CI/CD автоматизує процес розгортання. Функціональні та UX-тести підтвердили стабільність і зручність інтерфейсу, а економічний аналіз засвідчив окупність проекту за 3–4 місяці при аудиторії 4000 активних користувачів.

Робота містить 112 аркушів, 2 додатки, 9 таблиць, 18 рисунків і 25 джерел.

Ключові слова: Telegram-бот, Unity, клікер-гра, Spring Boot, Supabase, Docker, REST API, гейміфікація, монетизація.

SUMMARY

Kozar O.A. Development of a Telegram bot for gamified interaction with users: concept, architecture and implementation.

Bachelor's Qualification Thesis in the specialty 122 "Computer Science", educational program "Information Control Systems and Technologies". – Kyiv National University of Construction and Architecture. – Kyiv, 2025.

This thesis presents the development of an interactive Telegram bot, "Happy Dog Bot," which combines a Unity-based clicker game with the Telegram messenger to increase user engagement and create innovative monetization channels through an in-app news system and leaderboards showcasing the most active players.

The technical solution follows a client–server architecture: the Unity client (C#) uses coroutines for asynchronous HTTP requests to a Java/Spring Boot REST API with authentication and logging; PostgreSQL (Supabase) ensures reliable data processing and storage; and Docker with CI/CD automates deployment. Functional and UX tests confirmed interface stability and usability, while economic analysis demonstrated a payback period of 3–4 months with an audience of 4,000 active users.

The thesis comprises 115 pages, 2 appendices, 9 tables, 18 figures, and 25 references.

Keywords: Telegram bot, Unity, clicker game, Spring Boot, Supabase, Docker, REST API, gamification, monetization

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- **API** (Application Programming Interface) — інтерфейс програмування застосунків, набір методів взаємодії між клієнтом і сервером.
- **C#** (C Sharp) — об’єктно-орієнтована мова програмування, використана в Unity.
- **CI/CD** (Continuous Integration / Continuous Deployment) — безперервна інтеграція та безперервне розгортання, процес автоматичного тестування та деплою.
- **DAO** (Data Access Object) — шар доступу до даних, об’єкт для роботи з базою через ORM.
- **GDPR** (General Data Protection Regulation) — Загальний регламент захисту даних ЄС, який впливає на обробку персональних даних.
- **HTTP** (HyperText Transfer Protocol) — протокол передачі гіпертексту, основа REST-запитів.
- **HTTPS** (HTTP Secure) — захищений протокол HTTP із шифруванням SSL/TLS.
- **JSON** (JavaScript Object Notation) — легкий формат обміну даними між клієнтом і сервером.
- **MVP** (Minimum Viable Product) — мінімально життєздатний продукт, базова версія з первинним функціоналом.
- **MySQL / PostgreSQL** — реляційні СУБД, в роботі використовувалися MySQL (на Railway) та Supabase (PostgreSQL-сумісне).
- **MVC** (Model-View-Controller) — архітектурний патерн розмежування логіки, представлення та контролю.
- **OOP** (Object-Oriented Programming) — об’єктно-орієнтоване програмування, принцип розробки з акцентом на класи та об’єкти.
- **ORM** (Object-Relational Mapping) — технологія відображення об’єктів мов програмування у реляційну базу даних (Hibernate/JPA).
- **REST** (Representational State Transfer) — архітектурний стиль для створення веб-сервісів через стандартизовані HTTP-методи.

- **RPC / WebSocket** — WebSocket-протокол для двостороннього зв’язку в реальному часі між Unity та сервером.
- **SLA** (Service Level Agreement) — угода про рівень обслуговування, гарантія доступності хмарного провайдера.
- **SSL** (Secure Sockets Layer) — протокол безпечної передачі даних через мережу.
- **Swagger** — інструмент для документування REST-API та його інтерактивного тестування.
- **UML** (Unified Modeling Language) — уніфікована мова моделювання систем, діаграми класів, послідовності, варіантів використання.
- **UI** (User Interface) — інтерфейс користувача, візуальна частина програми.
- **UX** (User Experience) — досвід користувача, сукупність вражень від взаємодії з продуктом.
- **URL** (Uniform Resource Locator) — уніфікатор ресурсу, адреса ендпоінта чи веб-сторінки.
- **WebApp** — веб-додаток, в нашому випадку Unity-гра в Telegram WebApp.
- **WebGL** (Web Graphics Library) — API рендерингу 2D/3D-графіки у браузері, застосовано для Unity WebGL.
- **XP** (Experience Points) — очки досвіду в рамках гейміфікації.
- **QA** (Quality Assurance) — забезпечення якості, процес модульного, інтеграційного та системного тестування.

Одиниці та символи

- **₴** — українська гривня (UAH), валюта розрахунків вартості робіт і витрат.
- **$** — умовна одиниця внутрішньої валюти в грі (Score).
- **год** — години — одиниця обсягу трудовитрат.
- **міс** — місяці — одиниця часу для розрахунку окупності проекту.

ВСТУП

У сучасних умовах цифрової трансформації особливого значення набувають інноваційні інструменти комунікації між системами та користувачами. Одним із таких ефективних засобів є чат-боти, які активно застосовуються в різних сферах: освіті, бізнесі, охороні здоров'я, технічній підтримці та розвагах. Серед численних платформ для створення ботів особливою популярністю користується месенджер Telegram завдяки своїй відкритій API, високому рівню безпеки та широким можливостям налаштування.

Паралельно з розвитком чат-ботів зростає зацікавленість у використанні елементів гейміфікації — підходу, що передбачає інтеграцію ігрових механік у неігрові контексти задля підвищення мотивації, залученості та ефективності взаємодії. Гейміфікація активно використовується для стимулювання навчання, поліпшення сервісів і підвищення лояльності користувачів. Гейміфікація — це використання елементів ігрового дизайну в неігровому контексті для збільшення зацікавленості [22]. В освіті гейміфікація підвищує мотивацію, участь та закріплення знань серед учасників [25]

Актуальність теми обумовлена потребою в ефективних та інтерактивних інструментах, які дозволяють створити позитивний користувацький досвід, посилити зворотний зв'язок і мотивувати до регулярної взаємодії з цифровим продуктом. Telegram-боти з елементами гейміфікації мають значний потенціал у цьому контексті. Мотиваційні інформаційні системи інтегрують гейміфікацію для покращення користувацького досвіду й стимулювання бажаної поведінки [24].

Метою даної роботи є розробка Telegram-бота, орієнтованого на гейміфіковану взаємодію з користувачами. Для досягнення поставленої мети необхідно проаналізувати концепцію гейміфікації, визначити архітектурні особливості таких ботів, розробити та реалізувати відповідне програмне рішення з урахуванням функціональних і нефункціональних вимог.

Завдання дослідження:

1. Аналіз існуючих Telegram-ботів та підходів до реалізації гейміфікованої взаємодії з користувачами.
2. Проєктування архітектури Telegram-бота відповідно до функціональних вимог.
3. Розробка основних функціональних модулів Telegram-бота, включаючи елементи гейміфікації.
4. Впровадження та тестування Telegram-бота у тестовому середовищі або реальних умовах.
5. Оцінка ефективності гейміфікованої взаємодії та зручності використання розробленого Telegram-бота.

Об’єкт дослідження – процес взаємодії користувача з Telegram-ботом.

Предмет дослідження – Telegram-бот для гейміфікованої взаємодії з користувачами, його архітектура, функціональність та ефективність застосування гейміфікаційних елементів.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

У сучасному цифровому середовищі месенджери відіграють важливу роль у забезпеченні швидкої та зручної комунікації між користувачами. Одним із найпопулярніших сервісів є Telegram, який завдяки відкритому API та підтримці ботів дозволяє реалізовувати автоматизовану взаємодію у вигляді спеціалізованих сервісів. Telegram-боти активно застосовуються у сферах освіти, обслуговування клієнтів, маркетингу, розваг та багатьох інших галузях.

З метою підвищення інтересу та залучення користувачів до тривалої взаємодії з ботом активно застосовується підхід гейміфікації. Гейміфікація (gamification) — це використання ігрових елементів, таких як бали, досягнення, рівні, рейтинги, в контекстах, що не є іграми, з метою мотивації користувачів до виконання певних дій або поведінки. Застосування гейміфікації в ботах дозволяє підвищити зацікавленість, покращити користувацький досвід та сприяти досягненню поставлених цілей (наприклад, навчання, збору даних або просування продукту). Елементи гейміфікації (бали, бейджі, рейтинги) активують внутрішню мотивацію користувачів і підвищують їхню залученість [20]. Баланс між викликом і винагородою створює відчуття автономності, майстерності та мети в ігровому процесі [21].

На ринку представлено велику кількість Telegram-ботів, однак не всі з них використовують гейміфікаційні механізми. Крім того, ті, що мають такі елементи, часто обмежуються базовими функціями, не враховуючи індивідуальні потреби користувачів та не забезпечуючи достатньо гнучкої системи зворотного зв’язку чи інтерактивності.

З огляду на це, виникає потреба у розробці Telegram-бота, що поєднує функціональність, простоту використання та ефективну систему гейміфікованої взаємодії. Такий бот може мати широке практичне

застосування, зокрема у проєктах, де важливо підтримувати активність користувачів, стимулювати регулярну взаємодію або навчання.

1.1 **Постановка та аналіз проблеми**

У сучасних умовах цифрової трансформації особливої актуальності набувають автоматизовані сервіси, що забезпечують взаємодію з користувачами без участі людини-оператора. Одним із найпоширеніших інструментів у цьому напрямі є Telegram-боти. Вони дозволяють створювати інтерактивні системи для інформування, опитування, навчання, реєстрації, технічної підтримки тощо. Telegram має відкрите API, що надає широкі можливості для кастомізації функціоналу ботів та інтеграції з іншими системами.

Попри велику кількість Telegram-ботів, значна частина з них залишається однотипною та обмеженою у функціоналі. Часто вони виконують лише базові задачі, не враховуючи потреби довготривалої взаємодії або утримання уваги користувача. Такий підхід знижує ефективність цифрового продукту, адже користувачі швидко втрачають інтерес і припиняють взаємодію з ботом.

Одним із перспективних рішень цієї проблеми є впровадження гейміфікації — процесу використання ігрових механік у неігрових контекстах. Гейміфіковані системи стимулюють користувачів до регулярної взаємодії, створюють відчуття прогресу, змагання, досягнення цілей, що у підсумку підвищує залученість і лояльність до продукту. Особливо це актуально в освітніх, соціальних, корпоративних або інформаційних проєктах, де важливо підтримувати постійний зворотний зв’язок і мотивацію.

Однак створення ефективного гейміфікованого Telegram-бота вимагає комплексного підходу. Проблемними питаннями при розробці таких систем є:

1. відсутність чіткої методики інтеграції гейміфікаційних елементів у структуру Telegram-бота;
2. складність побудови інтуїтивного інтерфейсу у форматі текстової взаємодії;

3. обмеження Telegram API у порівнянні з повноцінними мобільними чи веб-застосунками;
4. необхідність забезпечення масштабованості та надійної архітектури при зростанні кількості користувачів;
5. відсутність системного аналізу ефективності гейміфікації у таких ботах.

Зважаючи на зазначене, постає задача розробки Telegram-бота, що поєднує функціональність, доступність і продуману систему гейміфікації. Такий підхід дозволить не лише вирішити проблему короткотривалої взаємодії, а й створити інструмент, що сприятиме досягненню довготривалих цілей проєкту — інформування, навчання, збору даних тощо.

Таким чином, сформульовано проблему: існує потреба у створенні Telegram-бота з інтерактивною та мотивуючою системою взаємодії, яка базується на гейміфікації, з урахуванням архітектурних, функціональних та UX-вимог.

1.2 Дерево цілей

Дерево цілей — це ієрархічна структура, яка відображає головну ціль проєкту, а також підцілі, що деталізують і конкретизують шлях до її досягнення. Такий підхід дозволяє чітко сформулювати задачі, визначити логіку розробки системи та розподілити етапи реалізації.

Основна ціль дипломної роботи – розробка та впровадження Telegram-бота для гейміфікованої взаємодії з користувачами, що передбачає високий рівень залучення, інтуїтивну взаємодію, надійну архітектуру, адаптивність до змін та ефективну реалізацію гейміфікаційних механік.

Для досягнення основної мети виділено такі ключові напрями:

1. Функціональність та зручність використання: бот має надавати логічну структуру команд, швидкий доступ до основних функцій, інформативні повідомлення, а також простий і зрозумілий інтерфейс взаємодії через чат.

2. Залучення користувачів через гейміфікацію: необхідно реалізувати систему балів, досягнень, рівнів або іншої мотиваційної механіки, яка стимулює регулярну активність, змагання та виконання певних завдань.
3. Надійність та безперебійна робота: Telegram-бот має функціонувати стабільно за будь-якого навантаження, забезпечувати збереження даних, мати логування подій, системи резервного копіювання та можливість масштабування при зростанні кількості користувачів.
4. Гнучкість та розширюваність: архітектура системи повинна дозволяти легке додавання нових функцій, адаптацію до змін у потребах цільової аудиторії, а також інтеграцію з зовнішніми сервісами або базами даних.
5. Оцінка ефективності гейміфікації: після впровадження Telegram-бота необхідно провести аналітику поведінки користувачів, зібрати відгуки, визначити рівень залученості та виявити елементи, які найбільше впливають на мотивацію до взаємодії.

1.3 **Вимоги та особливості проєктування системи**

Вимоги та особливості проєктування Telegram-бота є ключовим етапом у розробці системи, яка забезпечує ефективну гейміфіковану взаємодію з користувачами. Правильне визначення функціональних та нефункціональних вимог дозволяє створити зручний, безпечний та розширюваний інструмент, що відповідає сучасним технологічним стандартам.

Функціональні вимоги

- Реєстрація користувачів через Telegram (із збереженням базової інформації).
- Ідентифікація користувачів на основі Telegram ID.
- Система рівнів, балів або досягнень для стимулювання активності (гейміфікація).

- Надсилання push-сповіщень (повідомлень у чаті) з нагадуваннями, новинами та завданнями.
- Збір статистики про взаємодії користувача з ботом (баланс, активність, виконані завдання).

Безпека та конфіденційність

- Зберігання даних користувача в захищеній базі даних.
- Обмеження доступу до адміністративного функціоналу через автентифікацію.
- Обробка лише необхідної інформації згідно з принципом мінімізації даних.
- Надійність взаємодії між Telegram API та серверною частиною бота (використання SSL/HTTPS).

Інтерфейс взаємодії

- Використання кнопок Telegram (inline-кнопки, reply-клавіатура) для простоти навігації.
- Мінімалістичний текстовий інтерфейс, адаптований для мобільного користування.
- Можливість підтримки мультимовності (зокрема української та англійської мов).
- Відповіді бота повинні бути стислими, інформативними та приємними для сприйняття.

Інновації та технології

- Впровадження гейміфікаційних механік (система XP, рівнів, бейджів).
- Інтеграція з Google Sheets, Airtable або подібними інструментами для ведення статистики.
- Можливість зберігати дані в хмарних сервісах (Firebase, Supabase).
- Подальша інтеграція з AI або LLM-моделями для інтелектуального спілкування (опційно).

Особливості проєктування системи

Модульність

Архітектура Telegram-бота повинна бути побудована за принципом модульності — кожна функція реалізується як окремий компонент, що полегшує масштабування та супровід.

Розширюваність

Система має бути спроєктована з урахуванням можливості додавання нових ігрових механік, логіки завдань або API-інтеграцій без необхідності переписувати основну логіку.

Масштабованість

Telegram-бот має підтримувати роботу з великою кількістю одночасних користувачів, з урахуванням обмежень Telegram API.

Висока продуктивність

Забезпечення швидкого реагування на запити користувача. Використання кешування там, де це доцільно, для зменшення навантаження на сервер.

Логування та моніторинг

Система повинна фіксувати дії користувачів та внутрішні події для подальшої аналітики, усунення помилок та покращення взаємодії.

Гнучкість налаштувань

Можливість редагувати тексти повідомлень, завдання, параметри нагород тощо без втручання у код (наприклад, через Google Sheets або адмін-інтерфейс).

Автоматизація оновлень

Механізм оновлення сценаріїв або завдань без необхідності перезапуску бота. Це може бути реалізовано через періодичну синхронізацію з базою завдань або таблицею даних.

1.4 **Аналіз готових рішень**

Для успішної розробки Telegram-бота доцільно провести аналіз вже існуючих рішень, які реалізують схожі функції або використовують подібні підходи до взаємодії з користувачами. Аналіз таких ботів дозволяє не лише оцінити конкурентне середовище, але й виявити ефективні практики, які можна застосувати у власному проєкті. Розгляд переваг та недоліків дозволяє визначити ключові функціональні елементи, що приваблюють користувачів, а також слабкі місця, які варто врахувати для покращення власного рішення.

1.4.1 Hamster Kombat

Hamster Kombat (рис. 1.1) — це Telegram-бот, що набув популярності у 2024 році завдяки поєднанню простих ігрових механік, економічної стратегії та вірусного маркетингу. Його основна ідея полягає у створенні віртуального фінансового симулятора, де користувач виступає в ролі менеджера криптовалютної біржі, покращує її функціональність, купує апгрейди та заробляє внутрішню валюту.

Рисунок 1.1 – Hamster Kombat

Hamster Kombat використовує класичні елементи гейміфікації: рівні, місії, апгрейди, щоденні завдання, нагороди та можливість запрошувати друзів (реферальна система). Бот має інтуїтивно зрозумілий інтерфейс у форматі кнопок і повідомлень, що робить його зручним у використанні навіть для недосвідчених користувачів.

Основні функціональні особливості:

- Ігрова економіка з балансом, апгрейдами та доходами.
- Щоденні місії та завдання.
- Інтегрована реферальна система для залучення нових користувачів.
- Автоматичне нарахування внутрішньої валюти.
- Рейтингова система для підвищення залученості.

Сильні сторони:

- **Проста та захоплива механіка**, яка не потребує багато часу на навчання.
- **Вірусна модель поширення** через Telegram та соціальні мережі.
- **Масштабованість**: бот здатний обслуговувати мільйони користувачів одночасно.
- **Потужний інтерес аудиторії** до тематики криптовалют, що підтримує високий рівень залученості.

Слабкі сторони:

- **Одноманітність контенту** після досягнення певного прогресу.
- **Відсутність персоналізації** та взаємодії між користувачами.
- **Мінімальна навчальна або практична цінність** — переважно розважальний інструмент.

1.4.2 LevelUp Life Bot

LevelUp Life Bot (рис.1.2) — це Telegram-бот, що допомагає користувачам ставити особисті цілі, ділити їх на завдання, виконувати ці завдання та отримувати за них досвід, рівні й досягнення. Основна мета — мотивація до розвитку шляхом ігрових механік.

Рисунок 1.2 – Hamster Kombat

Користувачі можуть додавати власні завдання або обирати зі списку рекомендованих. Виконання завдань нагороджується віртуальними балами досвіду (XP), валютами, досягненнями та відкриттям нових рівнів. Таким чином, бот сприяє розвитку навичок, самодисципліни і послідовного досягнення цілей.

Сильні сторони:

- Корисність у реальному житті — користувачі досягають реальних результатів.
- Гейміфікація особистої ефективності: бот мотивує до регулярної активності.
- Простий і зручний інтерфейс, який адаптований до Telegram.
- Гнучкість завдань — користувач сам обирає, над чим працювати.

Слабкі сторони:

- Менш приваблива візуальна частина, порівняно з ігровими ботами.
- Обмежена соціальна взаємодія — немає командної роботи або змагань.

- Відсутність інтеграції з зовнішніми трекерами (наприклад, Google Fit або календарями).

1.4.3 Duolingo Telegram Bot

Duolingo Telegram Bot (Рис. 1.3) — це бот, створений на основі функціоналу популярної платформи Duolingo для вивчення мов. У форматі Telegram-бота ця платформа дозволяє користувачам щодня виконувати короткі завдання з вивчення іноземної мови, отримувати за це бали досвіду, досягати нових рівнів, змагатися з друзями та отримувати нагороди.

Рисунок 1.3 - Duolingo Telegram Bot

Такий бот є прикладом **успішної гейміфікації навчального процесу** з використанням простих інтерактивних елементів, push-нотифікацій та психологічного ефекту “щоденних серій” (streaks).

Сильні сторони:

- Високий рівень залучення завдяки регулярним завданням і мотивації.
- Освітній ефект — корисність бота у щоденному житті.
- Мінімалістичний, але зручний інтерфейс у Telegram.
- Змагання з друзями та внутрішній рейтинг, що мотивує до постійної активності.

Слабкі сторони:

- Вузький функціонал — фокус лише на навчанні, без розширених інтеграцій.

- Може викликати втому при щоденному використанні, якщо не додавати нових типів завдань.
- Відсутність повноцінної мультимедійної підтримки (наприклад, відео або мікрофонні вправи, доступні в мобільному застосунку Duolingo).

1.4.4 Rose Bot

Rose Bot (Рис. 1.4) — це потужний модераторський Telegram-бот, який забезпечує керування групами та взаємодіє з користувачами через автоматичні команди, попередження, “рівні довіри” й гейміфіковані механізми участі. Застосовується переважно у великих спільнотах.

Рисунок 1.4 – Rose Bot

Бот реагує на поведінку користувачів: наприклад, учасники, які часто порушують правила, отримують попередження або автоматично блокуються. Водночас активні учасники можуть отримувати "бали репутації", спеціальні ролі або рівні доступу. Це створює гейміфіковане середовище, де поведінка користувача безпосередньо впливає на його статус у спільноті.

Сильні сторони:

- Високий рівень автоматизації взаємодії з користувачем.
- Гейміфікований контроль за поведінкою — попередження, репутація, ролі.
- Гнучкість налаштувань, що дозволяє адаптувати бота до будь-якої спільноти.

- Підтримка великих обсягів трафіку та учасників.

Слабкі сторони:

- Складність початкового налаштування — новим користувачам може бути важко розібратись.
- Менш “ігровий” вигляд, ніж у Hamster Kombat або Duolingo.
- Не сфокусований на індивідуальній взаємодії, більше підходить для груп.

1.5 **Постановка задачі**

Потрібно розробити Telegram-бота, який забезпечує гейміфіковану взаємодію з користувачами шляхом виконання щоденних завдань, отримання балів, відкриття рівнів та досягнень. Система повинна мотивувати користувача регулярно взаємодіяти з ботом, отримуючи при цьому позитивний досвід у вигляді віртуальних нагород, повідомлень і статусів.

Telegram-бот повинен бути доступним 24/7, реагувати на запити користувача, надавати статистику активності та дозволяти змагатися з іншими учасниками, формуючи рейтинг.

Вхідні дані:

- Інформація для реєстрації (ім’я, нікнейм, Telegram ID).
- Команди, надіслані користувачем (наприклад, /start, /task, /profile).
- Обрані відповіді на щоденні завдання.
- Дані активності (кількість взаємодій, виконані завдання).
- Запити для перегляду статистики або рейтингу.
- Повідомлення зворотного зв’язку або скарги.

Вихідні дані:

- Підтвердження реєстрації та привітальне повідомлення.
- Список щоденних/тижневих завдань.
- Показ профілю користувача (рівень, досвід, кількість виконаних завдань).
- Повідомлення про отримання нагород.

- Рейтинг користувачів за активністю.
- Нотіфікації (нагадування, повідомлення про нові завдання, оновлення).
- Звіт про загальну статистику використання бота.

Функції системи:

- Реєстрація користувача у базі бота при першому запуску.
- Видача щоденних/персоналізованих завдань користувачу.
- Нарахування балів та досвіду за виконання завдань.
- Присвоєння рівнів та статусів на основі активності.
- Перегляд профілю та статистики користувача.
- Перегляд загального рейтингу серед учасників.
- Повідомлення про досягнення, нагороди, нові функції.
- Система зворотного зв’язку (відгуки, пропозиції).
- Автоматичні нагадування для підтримки щоденної взаємодії.
- Адмін-панель для управління завданнями та контентом.

Більш детальний опис програми представлено в додатку А.

Розділ 2. ПРОЕКТ ПРОГРАМНОГО ЗАСТОСУНКУ

Під час реалізації даного проєкту я керувався принципами об’єктно-орієнтованого програмування (ООП) та модульного підходу до структурування гри. Хоча проєкт реалізовано не у веб-середовищі, а за допомогою рушія Unity, архітектурна логіка, яку я використовував, частково наслідує концепції MVC-підходу, зокрема у частині розмежування відповідальностей між логікою, візуальним представленням та обробкою даних.

Загалом, під час проєктування ігрової системи я виділив декілька основних функціональних модулів: екран основної взаємодії (тобто клікер), магазин апгрейдів, модуль новин, налаштування користувача, а також серверну частину з базою даних. Кожен із цих модулів було реалізовано як окремий блок з чітко визначеною логікою, а взаємодія між ними здійснювалася через добре визначені інтерфейси. Це дозволило не тільки уникнути зайвої залежності між частинами гри, а й забезпечило зручність у підтримці та майбутньому розширенні функціоналу.

Основний ігровий екран (клікер). Центральною частиною застосунку є екран клікер-гри, де гравець може взаємодіяти з ігровим об'єктом — собакою — шляхом натискання (кліку), за що він отримує віртуальні монети. Сам механізм натискання реалізовано за допомогою подій Unity (через систему OnPointerDown та OnPointerClick), а обробка кожного натискання відбувається через окремий скрипт, відповідальний за підрахунок монет та оновлення інтерфейсу в реальному часі.

Цей модуль був реалізований максимально ізольовано: уся логіка обрахунку, збереження кількості монет, а також обробка пасивного доходу зберігалася у відповідних класах моделі, які не мали прямого зв’язку з візуальними елементами. Таким чином, будь-яку зміну UI можна провести без втручання в логіку підрахунку ресурсів.

Магазин апгрейдів. Наступним важливим компонентом є магазин. У ньому користувач має змогу витрачати зароблені монети на покращення: збільшення кількості монет за клік, підвищення пасивного доходу тощо. Кожне з покращень має свою вартість, рівень, а також унікальний ефект. Ця логіка була реалізована за допомогою системи скриптових об’єктів Unity (ScriptableObject) — кожен тип апгрейду описаний як окремий клас із відповідними методами, що викликаються під час взаємодії.

Особливу увагу я приділив збереженню стану покращень — інформація про поточний прогрес користувача зберігається локально, з можливістю синхронізації з базою даних, якщо гравець має підключення до інтернету. Таким чином, навіть у разі перезапуску гри прогрес не втрачається.

Модуль налаштувань. Окремим екраном реалізовано налаштування, де користувач може вмикати або вимикати звук, а також змінювати інші параметри гри. Для цього використано системні API Unity, зокрема PlayerPrefs, що дозволяє зберігати базові параметри у постійну пам’ять. Налаштування також можуть бути розширені у майбутньому — наприклад, додавання темної теми або вибору мовного інтерфейсу. Важливо зазначити, що налаштування не впливають на ігрову логіку напряму, а лише на візуальні та поведінкові аспекти інтерфейсу.

Вкладка новин. Однією з особливостей застосунку стала реалізація вкладки з новинами. Інформація для цієї вкладки автоматично надходить із телеграм-каналу, а потім зберігається у базі даних MySQL, розгорнутій на хостингу Railway. Далі ці новини завантажуються до застосунку через HTTP-запити у форматі JSON, де вони розпаршуються та виводяться на екран. Таким чином, гравець завжди бачить актуальні новини або анонси — наприклад, про майбутні оновлення, івенти, знижки тощо.

Для парсингу новин із телеграму використовувався окремий скрипт який періодично перевіряє наявність нових публікацій і, за потреби, додає їх у базу даних. У Unity реалізовано окремий контролер, який щойно гравець відкриває вкладку новин, надсилає запит до бази та отримує потрібні записи.

Серверна частина та база даних. Серверна частина реалізована на основі MySQL бази даних, яка була розгорнута на безкоштовному хмарному сервісі Railway. У базі реалізовано щонайменше дві таблиці — users та news. Таблиця users зберігає базову інформацію про гравця (ідентифікатор, кількість монет, рівень покращень тощо), а таблиця news — всі отримані новини з телеграму.

Доступ до бази даних здійснюється через HTTP API, запити до якого відбуваються у фоновому режимі (асинхронно), що дозволяє не блокувати основний геймплей. Я використовував прості REST-запити GET/POST для обміну інформацією. Принципи REST–архітектури базуються на безстатевих запитах і кешованих відповідях для масштабованості та надійності систем [17].

Архітектурні підходи та ООР. Гра розроблялась з активним використанням об’єктно-орієнтованого підходу. Кожен компонент гри (клік, апгрейд, новини, гравець) було реалізовано як окремий клас з чітко визначеними полями і методами. Це дозволило уникнути дублювання коду та забезпечило гнучкість при розширенні функціоналу. Наприклад, додавання нового апгрейду вимагало лише створення нового скриптового об’єкта без необхідності змінювати основну логіку гри.

Проектування архітектури застосунку стало не менш важливим, ніж сама реалізація. У процесі я набув практичного досвіду в тому, як ефективно структурувати ігрову логіку, як організовувати взаємодію між компонентами та як забезпечити масштабованість навіть у межах такого на перший погляд простого жанру, як клікер.

UML-діаграми, які я також створював у процесі планування, допомогли мені краще усвідомити зв’язки між частинами системи. Було складено як мінімум діаграму класів, діаграму випадків використання та послідовностей. Це не тільки допомогло мені самому краще орієнтуватися в коді, але й може стати у пригоді іншим розробникам у разі передачі або масштабування проєкту.

2.1 Ескізний проект

На етапі ескізного проєкту було сформовано загальну концепцію ігрового застосунку, визначено основні компоненти інтерфейсу користувача, їх розташування та взаємозв’язки. Основна мета цього етапу — візуалізувати загальну структуру майбутньої гри, аби сформувати чітке бачення реалізації функціоналу ще до написання коду.

Головна ігрова сцена. Основний екран гри був спроєктований з урахуванням принципу максимального залучення користувача. У центрі сцени розміщується інтерактивне зображення собаки, натискання на яку запускає механіку клікеру та нарахування монет. У верхній частині екрану розміщується інформаційна панель, де відображається загальна кількість монет, поточний рівень гравця, а також швидкість пасивного доходу.

У нижній частині сцени передбачені кнопки швидкого доступу до інших вкладок: **магазину**, **налаштувань** та **новин**. Також зліва або справа можна розмістити додаткові елементи інтерфейсу — наприклад, індикатори досягнень або підказки.

Сцена магазину. Екран магазину складається зі списку доступних покращень, кожне з яких представлено у вигляді окремої картки з назвою, описом, поточним рівнем, вартістю та кнопкою покупки. Покращення поділено на дві категорії: активні (збільшення монет за клік) та пасивні (підвищення монет, що генеруються автоматично).

З ескізної точки зору магазин реалізований як вертикальний скролюваний список, адаптований під мобільні пристрої. Важливим елементом UX-дизайну було забезпечення наочності: користувач повинен одразу розуміти, яке покращення доступне, що воно робить та чи вистачає йому ресурсів на покупку.

Сцена налаштувань. Ескіз сцени налаштувань включає перемикачі (toggle) для увімкнення/вимкнення звуку, вибору мови інтерфейсу (опціонально) та скидання прогресу гри. У майбутньому можливе доповнення налаштувань параметрами графіки або сповіщень.

Кожен пункт має зрозумілий підпис, а розміщення кнопок відбувається за принципом вертикальної ієрархії — від найчастіше використовуваних до менш пріоритетних.

Сцена новин. Вкладка новин реалізована у вигляді стрічки з картками новин. Кожна картка містить заголовок новини, короткий опис та дату публікації. У разі потреби користувач може натиснути на новину та побачити її повний текст.

З ескізної точки зору сцена новин нагадує типову стрічку новин, знайому користувачам із соціальних мереж, що полегшує сприйняття інформації. Зверху розміщується заголовок "Новини гри", під яким відображається список актуальних повідомлень.

Структура переходів між сценами. Для зручності навігації всі основні сцени пов’язані між собою через систему кнопок, які викликають зміну активної сцени або панелі. У Unity ця логіка була реалізована через SceneManager або через Canvas-перемикачі, якщо сцени реалізовані у межах одного екрану.

2.1.1 Аналіз предметної області та постановка задачі

Перед початком безпосередньої розробки програмного застосунку було здійснено аналіз предметної області, який дозволив виявити ключові особливості, вимоги користувачів, а також поточні тенденції в сфері мобільних клікер-ігор, що останніми роками стали доволі популярним жанром.

Клікер-ігри (інколи звані "idle games" або "tap games") характеризуються простотою взаємодії з користувачем: основна механіка полягає в багаторазовому натисканні на об'єкт для отримання ігрової валюти, яка згодом може використовуватись для покращень, що, у свою чергу, прискорюють темп гри або додають нові можливості. Психологічна привабливість таких ігор ґрунтується на постійному відчутті прогресу, візуальній винагороді та елементах колекціонування.

Під час аналізу також було звернуто увагу на успішні приклади з реального ринку, зокрема на такі ігри, як *Hamster Kombat*, *Cookie Clicker*, *Adventure Capitalist*, *Tap Titans* тощо. Спільною рисою всіх цих продуктів є лаконічний геймплей, інтуїтивно зрозумілий інтерфейс, наявність системи пасивного доходу (коли валюта генерується навіть без дій користувача), поступове відкриття нових можливостей, а також додаткові інформаційні вкладки для занурення у світ гри (новини, персонажі, опис подій).

На основі цього було сформульовано основні вимоги до майбутнього застосунку:

- мінімалістичний і зручний інтерфейс, оптимізований під мобільні пристрої;
- наявність інтерактивної зони клікеру (в нашому випадку — зображення собаки, по якому слід натискати для отримання монет);
- система прокачування, яка включає покращення пасивного доходу та бонусів за натискання;
- налаштування, що дозволяють користувачу адаптувати гру під себе (звук, скидання прогресу тощо);
- стрічка новин, яка підтягується з бази даних і оновлюється на основі Telegram-паблiку;
- зв'язок із серверною частиною та базою даних для зберігання інформації про користувачів та новини;
- легка масштабованість системи у майбутньому (додавання нових вкладок, покращень, гейміфікаційних механік).

Таким чином, **метою** розробки стало створення мобільного застосунку у вигляді телеграм-подібної гри-клікера, що має просту, але затягуючу ігрову механіку, можливість прокачки, а також функцію інформаційної взаємодії через стрічку новин. Гра повинна бути реалізована за допомогою середовища Unity з подальшою інтеграцією з базою даних на MySQL, розміщеною на хмарному сервісі Railway.

На основі аналізу сформульовано **основні задачі**:

1. Розробити архітектуру гри з урахуванням об’єктно-орієнтованого підходу (ООП). Компонентна архітектура Unity дає змогу швидко створювати складну поведінку в іграх шляхом приєднання повторно-використовуваних скриптів до об’єктів [1].
2. Реалізувати основну ігрову сцену з механікою клікеру.
3. Створити магазин з можливістю купівлі покращень.
4. Розробити інтерфейс налаштувань із базовими функціями керування.
5. Здійснити підключення до MySQL-бази даних та реалізувати обмін інформацією через API.
6. Налагодити механізм завантаження новин із Telegram-пабліку до бази даних.
7. Реалізувати відображення новин у вигляді зручної стрічки у грі.
8. Забезпечити збереження прогресу користувача, масштабованість і мінімальне навантаження на пристрій.

2.1.2 Контекстна діаграма

Контекстна діаграма — це високорівневе уявлення інформаційної системи, яке показує взаємозв’язки між основною системою та зовнішніми суб’єктами (акторами), з якими вона взаємодіє. У контексті нашого проєкту клікер-гри, основною системою виступає **мобільний застосунок**, створений на Unity, а зовнішні суб’єкти — це **користувач**, **база даних**, **Telegram-бот або паблік**, а також **веб-сервер**, що опрацьовує запити (рис. 2.1).

Основні взаємодії виглядають так:

- **Користувач** взаємодіє з грою: клікає для заробітку, купує покращення, переглядає новини, змінює налаштування.
- **Гра** взаємодіє з **сервером**, надсилаючи запити до **бази даних Railway (MySQL)** для зчитування/збереження прогресу користувача та отримання новин.

- **Telegram-паблік** через зовнішній скрипт (поза межами гри) наповнює таблицю новин у базі даних.

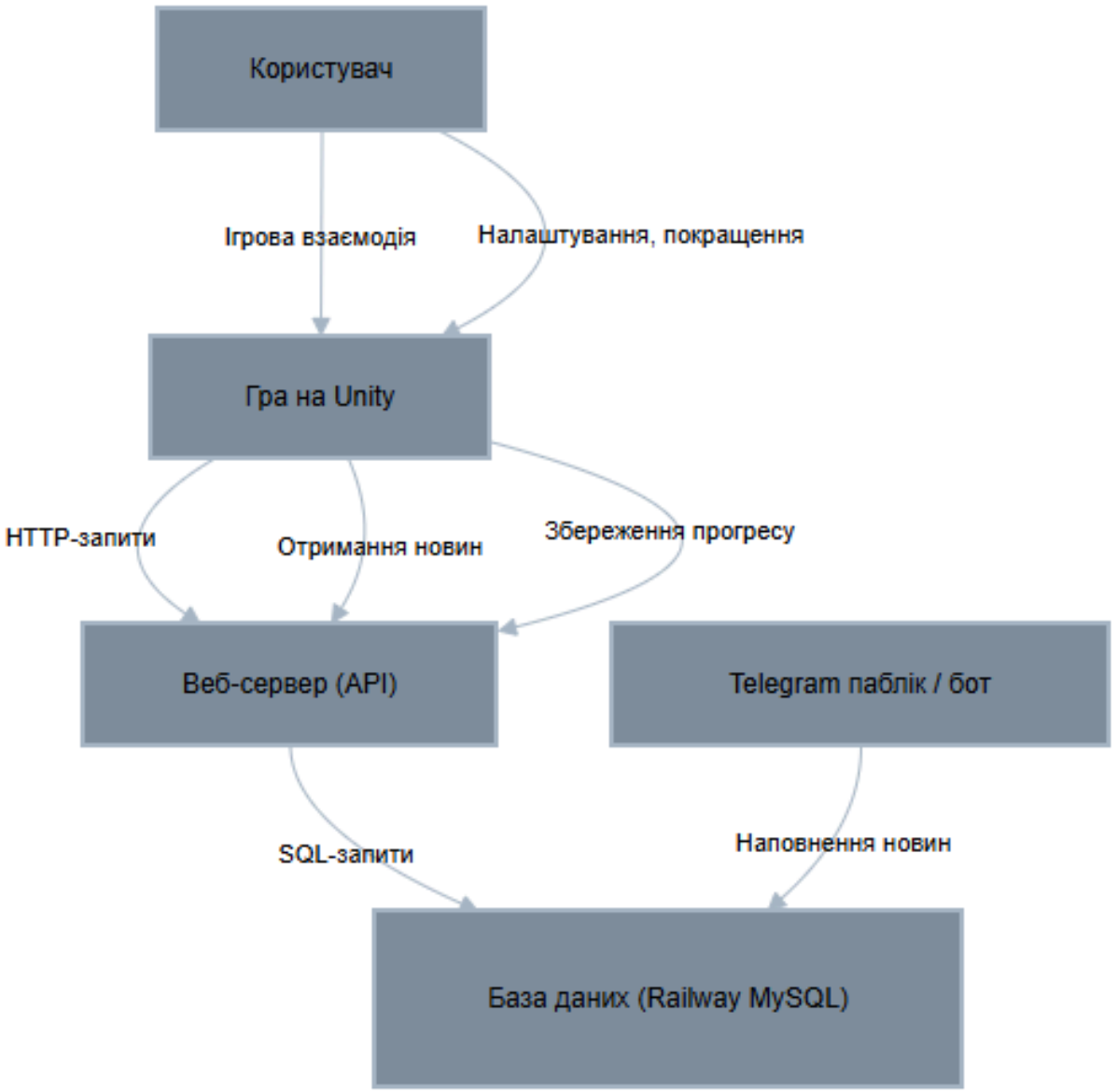

Рис. 2.1 – Контекстна діаграма

2.1.3 Діаграма варіантів використання

На цьому етапі розробки важливо було сформулювати основні сценарії взаємодії користувача з додатком, тобто визначити ті можливості, які гравець матиме в рамках створеного застосунку. Оскільки мова йде про гру клікерного типу, яка реалізована в середовищі Unity, то основна увага приділялась визначенню дій, пов'язаних із процесом гри, налаштуваннями, оновленням і отриманням інформації з бази даних.

Діаграма варіантів використання дозволяє наочно побачити ролі (акторів) у системі та ті функції, які їм доступні. У даному випадку головним і єдиним користувачем є гравець — звичайна людина, яка запускає застосунок на своєму пристрої. Він має змогу взаємодіяти з ігровим інтерфейсом, натискати на об'єкт для заробляння монет, відкривати магазин для покращення свого прогресу, переглядати вкладку з новинами, а також змінювати налаштування. Усі ці функціональні дії, хоча й здаються простими, потребують відповідної серверної логіки, обробки запитів, взаємодії з базою даних, а подекуди — й динамічного завантаження вмісту ззовні.

Окрім самого користувача, є й непряма взаємодія з Telegram-пабліком, звідки надходять новини. Ця частина є автоматизованою, і користувач її не бачить безпосередньо, проте результати — у вигляді новин у вкладці — стають частиною його досвіду. Відповідно, необхідно враховувати і цей момент під час проєктування варіантів використання. Кожен елемент на діаграмі відображає не просто кнопку чи дію, а частину загального процесу, який формує ігровий цикл, забезпечує збереження прогресу, синхронізацію даних та, загалом, — інтерактивність у межах гри.

Таким чином, сформована діаграма (рис.2.2) охоплює всі основні можливості гравця, дозволяючи побачити загальну логіку використання гри в цілісному вигляді. Вона стала для мене не лише засобом формалізації задуму, а й інструментом перевірки — чи не забуто якусь важливу частину функціоналу, чи логічно зв’язані між собою всі дії, чи немає зайвих або надмірно перевантажених сценаріїв.

2.1.4 Концептуальна модель застосунку

Під час розробки основним завданням було сформувати концептуальну модель, яка б охоплювала всі ключові сутності, зв’язки між ними та базову логіку взаємодії користувача із системою. На цьому етапі я намагався не занурюватися в технічні деталі реалізації, натомість зосередився на загальній логіці роботи додатку — саме це є суттю концептуального моделювання.

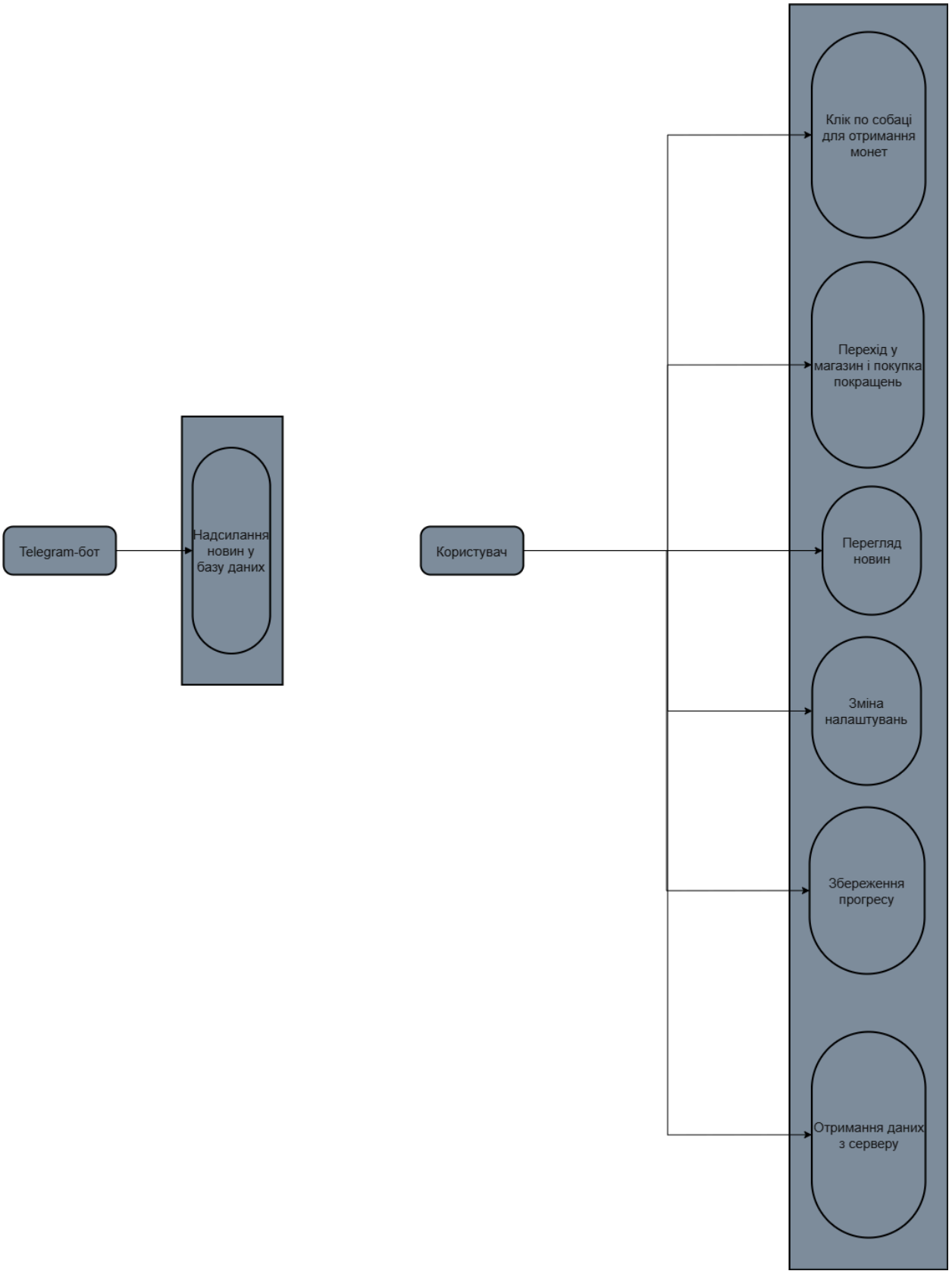

Рис. 2.2 – Use-case Діаграма

Модель було створено з урахуванням принципів об’єктно-орієнтованого підходу. Я виділив основні об'єкти, які беруть участь у взаємодії, та визначив, які атрибути і методи вони повинні мати. Таким чином, концептуальна модель стала відправною точкою для наступних етапів проєктування, дозволивши краще зрозуміти структуру системи ще до написання коду.
Ключові об’єкти в межах проекту представлені в таблиці 2.1.

Таблиця 2.1 - Основні об’єкти

Об’єкт	**Призначення**
Користувач	Основна сутність, для якої зберігається прогрес, кількість монет, рівень покращень тощо. Цей об’єкт також відповідає за налаштування (наприклад, звук) і взаємодію з інтерфейсом.
Гра	об’єкт, який керує логікою клікання, генерації пасивного доходу, а також виступає посередником між користувачем і системою.
Покращення	об’єкти, які можна придбати в магазині. Кожне з них впливає або на пасивний дохід, або на кількість монет, що нараховуються за клік.
Новини	сутність, яка наповнюється через Telegram-бота, зберігається в базі даних і відображається у відповідній вкладці додатку.
База даних	на концептуальному рівні — це місце зберігання всієї важливої інформації: дані користувача, список новин, стан гри. У реалізації використовується MySQL

Усі ці об'єкти пов’язані між собою у логічну структуру. Наприклад, кожен користувач має набір покращень, може читати новини, отримувати монети, а також зберігати і відновлювати свій прогрес із бази даних. Новини,

у свою чергу, оновлюються через окремий телеграм-бот, що передає інформацію на сервер, і потім ця інформація використовується всередині гри.

Це моделювання дало змогу зосередитися на логіці взаємодії між об’єктами, перш ніж заглиблюватися в технічну реалізацію. На практиці це допомогло уникнути багатьох помилок у майбутньому, а також зробило систему більш структурованою та гнучкою до подальших змін.

На наступному етапі проєктування на основі цієї моделі я вже міг створювати UML-діаграми класів (рис.2.3) і чітко розписувати методи, атрибути та зв’язки між об’єктами, спираючись на закладену логіку, яку вдалося викристалізувати саме завдяки концептуальній моделі.

2.1.5 Діаграма станів бота

Оскільки бот виконує вузьке, але важливе завдання, його поведінка досить лінійна та передбачувана, що дозволило зручно змоделювати її за допомогою діаграми станів (рис.2.4).

Основний стан, у якому перебуває бот — це очікування новин. У цей момент він перевіряє наявність нових публікацій у телеграм-каналі. Як тільки з’являється новий пост, система переходить у стан отримання новин з пабліку, після чого запускається процедура обробки новини — вилучення тексту, фільтрація небажаного контенту, очищення та, за потреби, скорочення чи форматування повідомлення. Якщо новина відповідає критеріям валідності, відбувається надсилання в базу даних, після чого бот знову повертається до пасивного стану очікування.

Варто зазначити, що у випадку, якщо новина не відповідає умовам (наприклад, містить заборонені слова, надто коротка чи є дублікатом), система не виконує ніяких записів, а одразу переходить до стану очікування нової публікації. Такий підхід дозволяє уникнути засмічення бази даних та забезпечує якість контенту, що відображається безпосередньо у грі.

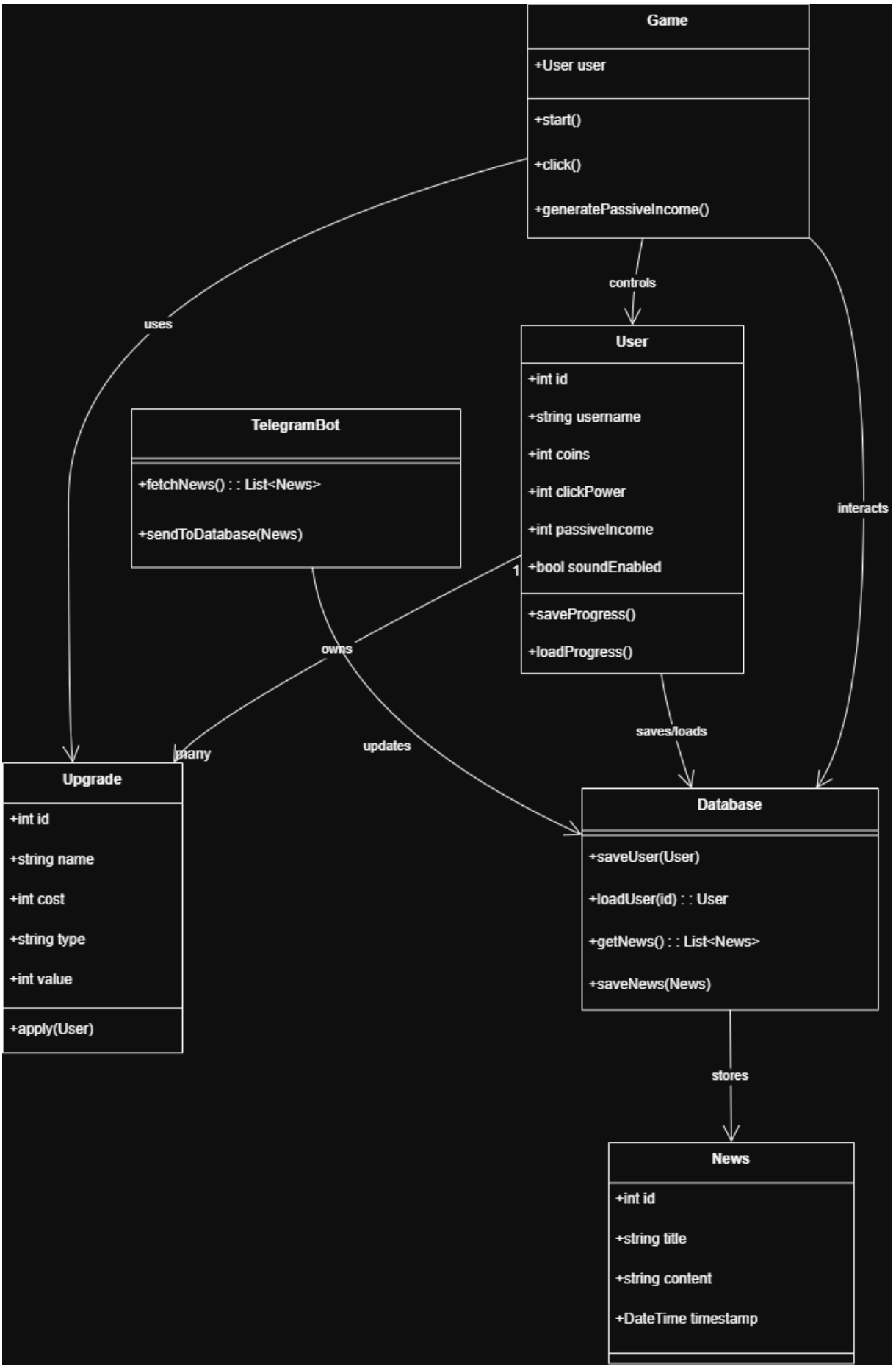

Рис. 2.3 – Діаграма класів

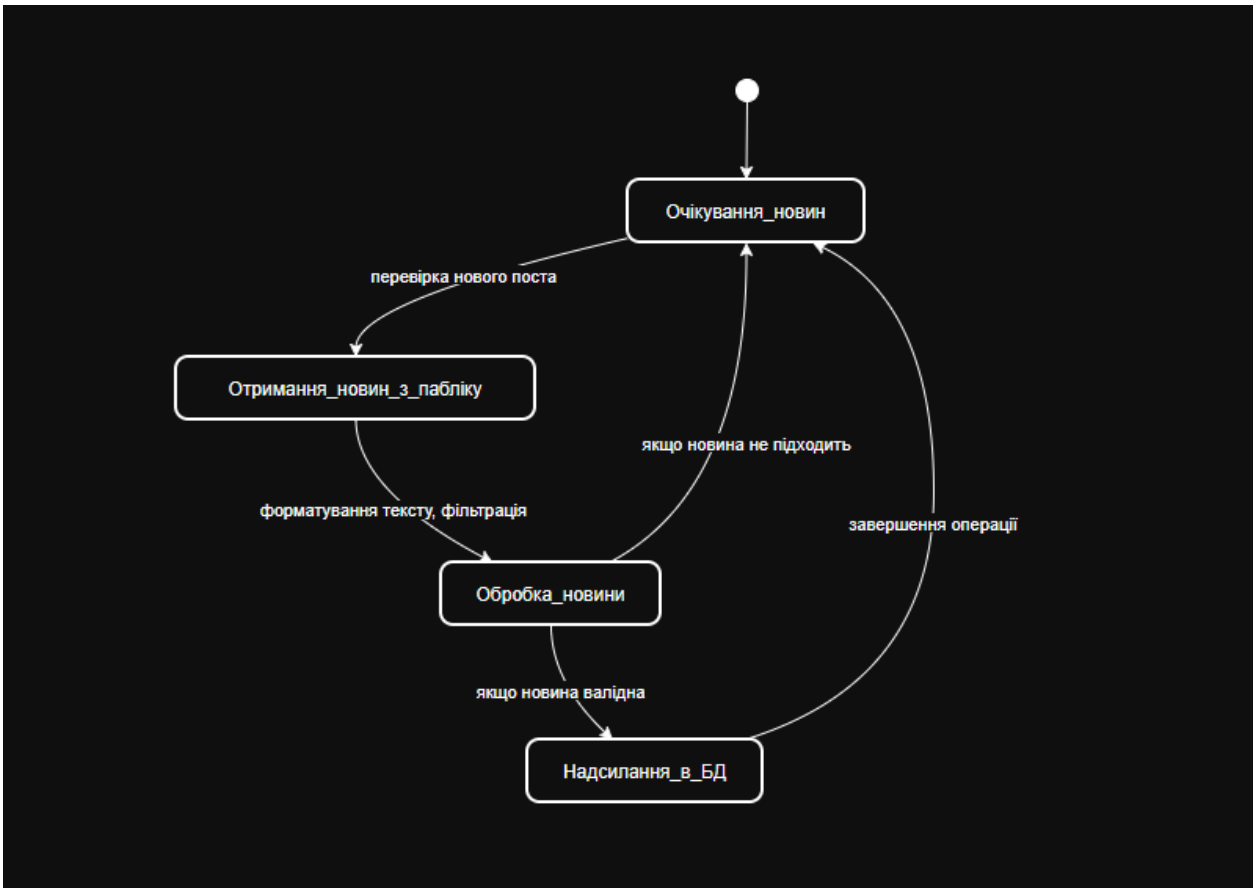

Рис. 2.4 – Діаграма станів

Розробка такої логіки на основі станів дозволила не лише чітко структурувати поведінку бота, а й забезпечити його стабільну роботу при тривалому розгортанні на сервері. В майбутньому така структура також спрощує масштабування, наприклад, для обробки кількох джерел одночасно або додавання нових фільтрів обробки новин.

2.1.6 Проектування користувацького інтерфейсу

На етапі проектування користувацького інтерфейсу особливу увагу приділено зручності взаємодії користувача з додатком, візуальній простоті та зрозумілій навігації. Оскільки гра належить до жанру клікерів, її основна мета — забезпечити просту, повторювану, але водночас візуально приємну ігрову механіку, яка б не вимагала додаткового навчання користувача (рис.2.5).

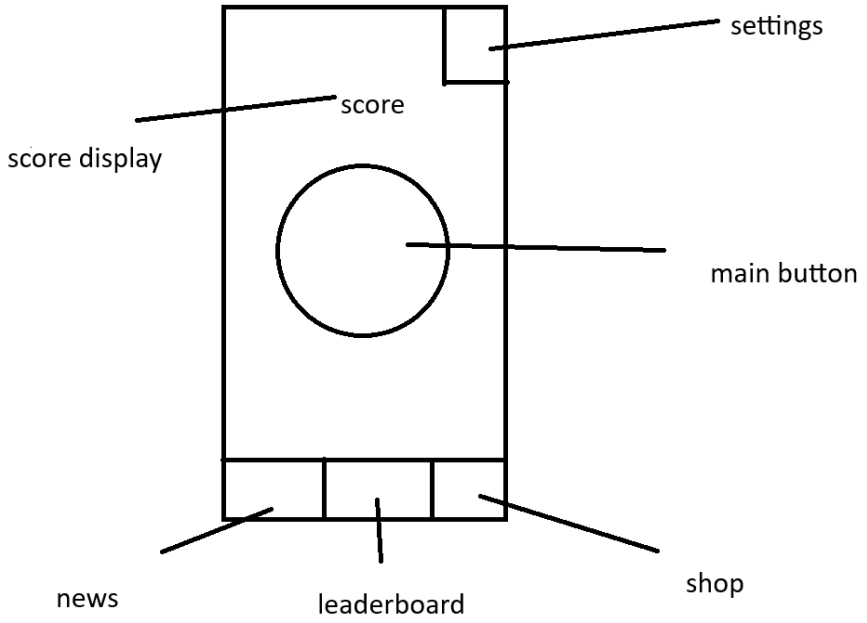

Рис. 2.5 – Макет головної сторінки

Головний екран було спроектовано з фокусом на центральний ігровий об’єкт — собаку, на яку можна натискати для отримання монет. Візуальне натискання супроводжується анімацією та звуком, що створює ефект задоволення від взаємодії. У верхній частині екрану виводиться кількість монет, які є у користувача, а також показники пасивного доходу. У нижній частині розміщене меню з іконками, що ведуть до основних вкладок:

Магазин, **Новини**, **Налаштування**. Вкладка **Магазин** реалізована у вигляді списку покращень, кожне з яких має назву, опис і вартість (рис.2.6). Користувач може купувати апгрейди, що підвищують кількість монет за клік або збільшують пасивний дохід. Покращення структуровано за категоріями, і дизайн було продумано таким чином, щоби вся взаємодія займала мінімум кліків — основні дії виконуються буквально одним натисканням.

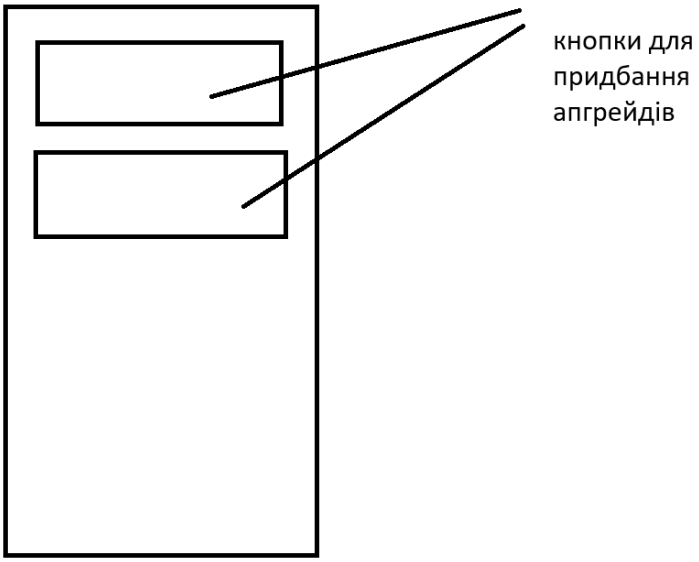

Рис. 2.6 – Макет магазину

Вкладка **Новини** — це важлива частина проєкту, яка відображає останні записи з телеграм-каналу, передані через базу даних. Дизайн цієї частини максимально схожий на стрічку, знайому користувачу за соціальними мережами: заголовок, дата, короткий текст. Це сприяє інтеграції гравця в інформаційне поле гри та дозволяє підтримувати актуальність контенту навіть за умови довготривалого користування.

Рис. 2.7 – Сторінки з новинами

Вкладка **Налаштування** дозволяє вмикати або вимикати звук, регулювати музику, а також скидати прогрес або синхронізувати дані. Для збереження простоти інтерфейсу було вирішено уникати складних систем реєстрації — замість цього використовується внутрішній профіль, пов’язаний з унікальним ID користувача.

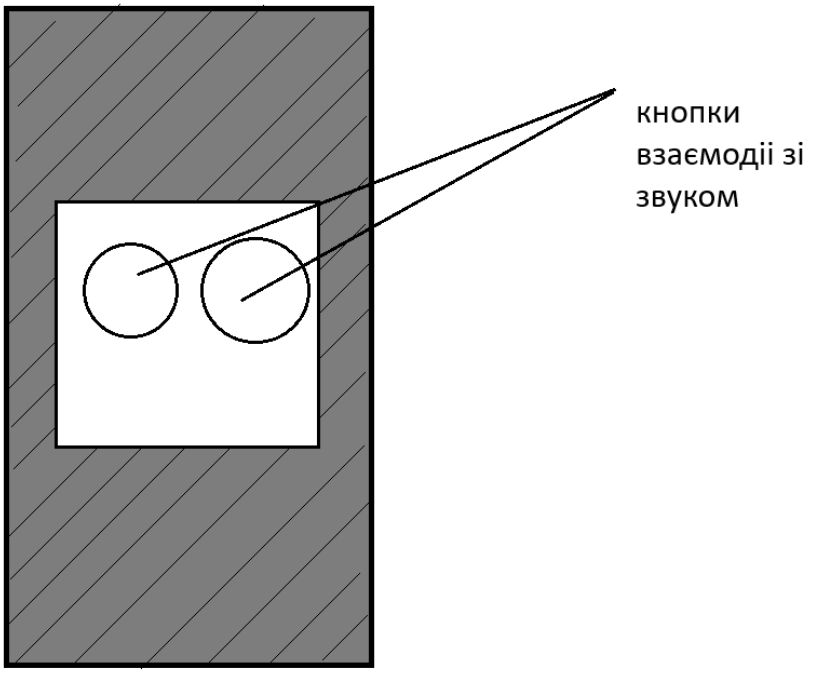

Рис. 2.8 – Сторінки з налаштуванням

При розробці інтерфейсу активно використовувалися принципами UX-дизайну: мінімізація кількості дій для досягнення цілі, зрозумілість іконок, візуальна ієрархія, зручне розміщення елементів керування. Колірна гама була підібрана так, щоб гра виглядала сучасною, але не викликала втоми при тривалому користуванні.

Реалізація інтерфейсу здійснювалась у середовищі Unity з використанням UI Canvas, анімаційних компонентів та стандартних засобів компонування елементів (Grid, Vertical Layout, Anchors). Важливою складовою стало адаптивне розташування елементів — інтерфейс однаково зручно виглядає як на широких екранах, так і на смартфонах із різним співвідношенням сторін.

У цілому, проектування інтерфейсу було не лише технічним етапом, а й процесом, який дав мені глибше розуміння того, як важливо думати про користувача ще на ранніх етапах розробки. Саме зручність, логіка та простота взаємодії визначають успіх додатку незалежно від технічної складності його реалізації.

2.2 Технічний проект

У цьому розділі моєї дипломної роботи я детально розглядаю технічну реалізацію розробленого програмного продукту. Моя основна мета в цьому розділі — надати комплексний опис ключових аспектів системи, починаючи від її функціональних можливостей і архітектурної побудови, до структури та способів моделювання даних.

Технічний проект є основою для розуміння того, як я спроектував систему, як її компоненти взаємодіють між собою, та які технологічні рішення були мною обрані для досягнення поставлених цілей. У роботі я представляю опис архітектури, що включає клієнтську частину (гра, розроблена на платформі Unity, та інтерфейс у месенджері Telegram), серверну частину (API-сервер, реалізований на мові Java та запущений у Docker-контейнері) та систему управління базами даних, яка також розгорнута в ізольованому контейнері.

Особливу увагу я приділив механізму взаємодії між ігровим клієнтом та сервером, зокрема використанню корутин (coroutines) у Unity для асинхронної відправки запитів на збереження та отримання даних, що, на мою думку, є ключовим для забезпечення безперервності ігрового процесу. Короутини в Unity дозволяють виконувати логіку протягом кількох кадрів без блокування головного потоку, що корисно для затриманих дій та оновлень рахунку [2].

Подальші підрозділи послідовно розкривають такі теми:

- **Функціональні можливості системи:** Опис основних функцій, доступних кінцевому користувачеві в Telegram-боті та в інтегрованій грі.

- **Архітектурна побудова програмного забезпечення:** Детальний розгляд обраної архітектурної моделі, обґрунтування використання мікросервісного підходу з контейнеризацією за допомогою Docker, а також опис взаємозв'язків між усіма компонентами системи.
- **Структура та моделювання даних:** Аналіз структури даних, що зберігаються в базі даних, опис сутностей та зв'язків між ними.

2.2.1 Функціональні можливості системи

Розроблена система являє собою Telegram-бот з інтегрованою міні-грою, реалізованою на ігровому рушії Unity. Основна мета системи — забезпечити гейміфіковану взаємодію з користувачем через інтерфейс Telegram, що поєднує елементи гри, соціального змагання та персоналізації. Нижче наведено детальний опис функціональних можливостей, які забезпечує система.

1. Ініціалізація користувача

При першому запуску Telegram-бота система автоматично реєструє користувача в базі даних. Це відбувається шляхом передачі Telegram ID та username, які є унікальними ідентифікаторами, що використовуються для ідентифікації гравця, прив’язки прогресу та подальшої персоналізації.

2. Гейміфікована механіка (Clicker Game)

- **Основна кнопка**: Основна ігрова активність реалізована у вигляді кліку по кнопці. Кожне натискання генерує певну кількість балів (Score), які надалі використовуються для розвитку профілю гравця.
- **Механіка накопичення**: З кожним кліком гравець заробляє ресурси, які можуть бути витрачені на вдосконалення параметрів.
- **Бонуси і покращення**:
  - **Покращення сили кліку (Click Buster)** — дозволяє збільшити кількість балів за одне натискання.
  - **Пасивний дохід (Passive Income)** — генерує бали автоматично з певною періодичністю (кожну секунду), навіть без дій користувача.

3. Панель магазину

У спеціальному розділі гри реалізована система апгрейдів:

- Гравець має змогу купувати покращення для кліку та пасивного доходу.
- Кожне наступне покращення коштує дорожче, що створює ігровий баланс.
- Усі покупки зберігаються у базі даних та синхронізуються з сервером.

4. Панель новин

Система включає інтеграцію з Telegram-ботом, що автоматично моніторить публічний Telegram-канал з новинами. Функціональність новин реалізована наступним чином:

- Зовнішній бот надсилає JSON-документ з останніми новинами.
- Unity-гра отримує цей документ, парсить його і відображає останні 5 новин на відповідній панелі.
- Це створює ефект живого контенту всередині гри.

5. Лідерборд (Таблиця лідерів)

Система реалізує змагальний елемент:

- Відображається список з 5 гравців, які набрали найбільшу кількість балів.
- Дані оновлюються періодично або після значних змін у прогресі гравця.
- Рейтинг базується на параметрі click_score.

6. Панель налаштувань

Для персоналізації досвіду гравця реалізована панель налаштувань:

- Увімкнення/вимкнення фонової музики.
- Увімкнення/вимкнення звукових ефектів.
- Усі зміни зберігаються та автоматично підтягуються при наступному запуску гри.

7. Збереження та відновлення прогресу

Оскільки вся інформація про користувача зберігається у централізованій базі даних:

- Прогрес не втрачається навіть після завершення сесії або повторного запуску Telegram-бота.
- Передача даних реалізована через API-запити до серверної частини.
- Unity використовує карутинові методи (IEnumerator), що дозволяють асинхронно надсилати та отримувати дані.

8. Інтеграція з Telegram

Користувач не потребує окремої реєстрації або входу — Telegram сам передає необхідні ідентифікаційні дані, що значно спрощує процес взаємодії:

- Навігація здійснюється безпосередньо в Telegram.
- Система здатна адаптуватися до кожного конкретного користувача.

Ці функціональні можливості забезпечують повноцінний гейміфікований досвід, який можна масштабувати, адаптувати під інші тематики або розширити за рахунок нових модулів (наприклад, PvP-режимів, челенджів, досягнень тощо). Усі компоненти тісно взаємодіють між собою, формуючи єдину екосистему, орієнтовану на залучення та утримання користувача. The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be serializable to go across the connection.[19]

2.2.2 Архітектурна побудова програмного забезпечення

Архітектура розробленої системи базується на модульному підході з чітким розділенням відповідальностей між компонентами. Це забезпечує масштабованість, гнучкість та спрощує супровід проєкту. Програмне забезпечення складається з трьох ключових рівнів: клієнтського (Unity), серверного (Java API) та рівня зберігання даних (БД у Docker-контейнері).

Загальна схема архітектури (Рис. 2.9)

1. Клієнтський рівень: Telegram-бот з Unity-грою

- **Unity-гра** вбудована в Telegram-бот як основний фронтенд.
- Вся логіка ігрового процесу (клікер, магазин, лідерборд, новини, налаштування) реалізована за допомогою скриптів на C#.

- Для асинхронної взаємодії з сервером використовуються **карутинові методи** (IEnumerator, UnityWebRequest) — це дозволяє без затримок отримувати і надсилати дані. UnityWebRequest надає інструменти для HTTP-запитів (GET, POST) із підтримкою JSON-пакетів і обробкою завантажень/відвантажень всередині гри [4].
- Кожна взаємодія (збереження прогресу, отримання лідерборду, завантаження новин) здійснюється через API-запити.

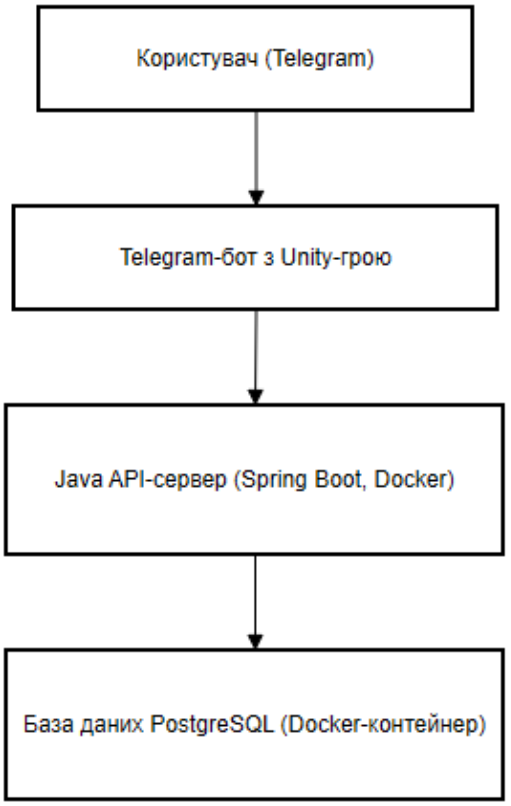

Рис. 2.9 – Загальна схема архітектури

Основні модулі Unity-клієнта:

- Game — обробка кліків, логіка ігрових балів, також інтегрована логіка магазину.
- ApiManager — єдиний модуль для взаємодії з API.
- LeaderboardManager — запити до лідерборду.
- NewsManager — обробка JSON-документів новин. Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.[18]

- SettingsManager — збереження та завантаження налаштувань.

2. Серверний рівень: API на Java (Spring Boot)
Серверна частина реалізована як RESTful API з використанням Java (Spring Boot), що надає безпечний та масштабований механізм взаємодії між клієнтом і базою даних.

Функції API-сервера:

- Отримання та реєстрація користувача за Telegram ID.
- Прийом та обробка запитів від Unity-клієнта (GET/POST).
- Збереження та оновлення ігрового прогресу.
- Надання JSON-даних про лідерів, новини та стан користувача.

Технологічні компоненти:

- Spring Boot (REST-контролери)
- Hibernate/JPA для ORM
- Maven як система складання
- базова аутентифікація (за Telegram ID)

Приклад API-ендпоінтів:

- POST /v1/users — реєстрація нового користувача
- PATCH /v1/users/{telegramId}/update — оновлення юзера через телеграм ID
- POST /v1/news — збереження нови в базу даних
- GET /v1/users/{telegramId} — отримання даних про гравця через телеграм ID
- GET /v1/users/top — отримання топ-5 гравців
- GET /v1/users/news — отриманняу новин

3. Рівень зберігання: база даних у Docker-контейнері
База даних розгорнута в ізольованому середовищі Docker-контейнера. Такий підхід гарантує:

- Високу переносимість системи.
- Простоту розгортання в будь-якому середовищі.
- Можливість масштабування.

Обрана СУБД: PostgreSQL

Зв'язок з API-сервером здійснюється через ORM (Hibernate), що дозволяє працювати з об'єктами Java замість сирих SQL-запитів.

Основні таблиці:

- users — Telegram ID, username, дата реєстрації.
- news — новинні повідомлення.
- settings – таблиця з налаштуваннями гри:

4. Docker-контейнери

Уся серверна інфраструктура працює в окремих Docker-контейнерах:

- unity-game-api — API-сервер.
- game-database — база даних.
- nginx-reverse-proxy — маршрутизація запитів.

Таке контейнеризоване рішення дозволяє:

- Швидко запускати або оновлювати будь-який компонент.
- Зменшити кількість залежностей при розгортанні.
- Забезпечити ізольованість середовищ.

5. Інтеграція з Telegram

Telegram API дозволяє отримувати унікальний ідентифікатор користувача (Telegram ID) та його username без необхідності додаткової авторизації. Ці дані використовуються для:

- Автоматичної реєстрації.
- Ідентифікації гравця при запитах.
- Прив’язки прогресу до конкретного користувача.

Архітектура програмного забезпечення розроблена з урахуванням принципів розширюваності, надійності та масштабованості. Завдяки поділу на логічні шари (клієнт → сервер → база), контейнеризації через Docker і використанню сучасних технологій (Unity, Spring Boot, REST, ORM), система здатна легко розвиватися в майбутньому, наприклад, шляхом додавання PvP-функціоналу, досягнень або внутрішніх подій.

2.2.3 Структура та моделювання даних

Система використовує реляційну базу даних **PostgreSQL**, розгорнуту в окремому контейнері Docker. Зв'язок з нею здійснюється через Java API-сервер за допомогою ORM-фреймворку Hibernate. Уся інформація про користувачів, їхній прогрес у грі, налаштування, новини та результати лідерборду зберігається у відповідних таблицях.

Основні сутності та зв’язки:

1. **Users** – основна таблиця, що містить інформацію про користувача Telegram:

- id (PRIMARY KEY) – унікальний ідентифікатор;
- telegram_id – Telegram ID користувача;
- username – ім'я користувача;
- settings_id – ідентифікатор налаштувань користувача
- score – рахунок користувача
- upgrade1 – рівень апгрейду множника кліку
- upgrade2 – рівень апгрейду множника пасивного доходу
- created_at – час створення запису про юзера
- updated_at – час останнього оновлення гри користувачем (треба для розрахунку пасивного доходу під час неактиву)

2. **Settings** – таблиця з налаштуваннями гри:

- id (PRIMARY KEY);
- sound – Bool значення яке зберігає стан налаштування звукових ефектів
- music – Bool значення яке зберігає стан налаштування фонової музики

3. **News** – таблиця для збереження новин, які надходять у вигляді JSON-документів:

- id (PRIMARY KEY);
- content – новина яку несе в собі повідомлення;
- created_at – час створення запису про створення запису в таблиці
- updated_at – час створення нового запису, для відображення тільки нових повідомлень

Загальні принципи моделювання:

- Усі зовнішні ключі підтримують цілісність даних через обмеження FOREIGN KEY.
- Доступ до бази даних реалізовано через репозиторії, де кожен репозиторій відповідає окремій сутності.
- Взаємодія між Unity-грою та API-сервером відбувається у вигляді HTTP-запитів, які повертають або приймають JSON-дані, що потім трансформуються у сутності для запису у PostgreSQL.
- Завдяки такій структурі забезпечується масштабованість, розширюваність і простота обслуговування системи.

2.3 Засоби реалізації та технічні рішення

Цей розділ присвячено обґрунтуванню вибору інструментів, технологій та архітектурних рішень, використаних під час реалізації телеграм-бота з інтегрованою Unity-грою. Зважаючи на складність та багаторівневість взаємодії між клієнтською частиною, сервером та базою даних, особливу увагу приділено питанням сумісності, масштабованості, продуктивності та зручності розробки.

Розділ структуровано на три ключові підпункти:

- 2.3.1 Обґрунтування вибору інструментарію — розкриває логіку вибору мов програмування, фреймворків, ігрового рушія, серверних рішень та бази даних;
- 2.3.2 Опис інтеграції Unity — деталізує процес взаємодії Unity-додатку з API-сервером, включаючи використання корутин, формування HTTP-запитів та обробку JSON-даних;
- 2.3.3 Тестування та відлагодження застосунку — описує методики тестування, виявлення й виправлення помилок, а також інструменти для діагностики та логування.

Ці підрозділи в сукупності демонструють повний технічний цикл створення системи, починаючи від вибору технологій і завершуючи забезпеченням її надійної роботи.

2.3.1 Обґрунтування вибору інструментарію

У процесі реалізації проєкту Telegram-бота з інтегрованою Unity-грою було обрано сучасний, гнучкий та перевірений технологічний стек, що забезпечує ефективну взаємодію між клієнтською частиною, сервером і базою даних. Нижче наведено обґрунтування вибору основних технологій.

1. Unity (C#)

Unity обрано як рушій для реалізації клієнтської частини гри через його:

- підтримку 2D-ігрових механік, необхідних для реалізації клікера;
- гнучку систему UI, що дозволяє створити магазини, лідерборд, новинну панель тощо;
- сумісність із мобільними платформами, зокрема Android, що дозволяє інтегрувати гру з Telegram через WebApp або запускати окремим застосунком;
- підтримку асинхронних викликів (через корутини), що дає змогу ефективно обробляти HTTP-запити до серверної частини без блокування головного потоку гри.

2. Telegram Bot API (через Java TelegramBots Library)

Цей API обрано для створення Telegram-бота завдяки. Telegram Bot API пропонує HTTP-інтерфейс для створення ботів із відправкою запитів через HTTPS і отриманням JSON-відповідей з оновленнями [5]:

- широкій документації та підтримці;
- можливості реалізації автоматизованих відповідей, взаємодії з WebApp або зовнішніми інтерфейсами;
- використанню TelegramBot Java Library, яка легко інтегрується з Spring Boot.

3. Java + Spring Boot (Back-end сервер)

Серверна частина реалізована на Java з використанням Spring Boot через:

- зручну архітектуру REST-контролерів. REST-архітектура визначає ресурси з унікальними URI й маніпуляцію ними через стандартизовані HTTP-методи [16];
- вбудовану підтримку ORM через Spring Data JPA Spring Data JPA надає абстракції репозиторіїв для простих CRUD-операцій і складних запитів без зайвого коду, використовуючи конвенції та анотації [8];
- можливість гнучкої обробки запитів Unity-клієнта та Telegram;
- підтримку високої масштабованості, що дозволяє адаптувати сервер до зростання навантаження.

4. PostgreSQL

Базу даних PostgreSQL обрано як надійне, продуктивне рішення для зберігання структурованих даних:

- підтримує реляційні зв’язки та транзакції;
- має розвинуту підтримку типів даних (JSON, масиви тощо);
- легко інтегрується зі Spring Data через JDBC та Hibernate.

5. Docker

Docker використано для ізоляції середовища виконання всіх компонентів системи Docker упакує додаток із усіма залежностями в контейнер, забезпечуючи однакову поведінку в різних середовищах (розробка, тест, продакшн) [10] Docker Compose спрощує опис і запуск мультиконтейнерних систем через YAML-конфігурацію командою в один клік [11].:

- дозволяє легко запускати сервер та базу даних у контейнерах;
- забезпечує портативність і незалежність від ОС розробника чи хостингу;
- дозволяє швидко розгортати оновлення або масштабувати систему.

6. JSON (формат обміну даними)

JSON використовується як основний формат обміну між клієнтом (Unity) і сервером. Java SE 17 вводить «record»-класи для компактного оголошення незмінних об’єктів-переносників даних (DTO) між клієнтом і сервером [9] JSON — легковаговий текстовий формат для обміну даними між клієнтами та серверами без зайвої розмітки [14]:

- підтримується більшістю мов програмування;
- легко парситься як на боці клієнта, так і сервера;
- ідеально підходить для передачі структурованих даних (прогрес гравця, новини, налаштування тощо).

2.3.2 Опис інтеграції Unity

Інтеграція ігрового клієнта, створеного на рушії Unity, з Telegram-ботом та серверною частиною проєкту є ключовим компонентом архітектури системи. Завдяки правильно реалізованому обміну даними між компонентами, гра працює синхронно з Telegram-ботом, забезпечуючи індивідуальну взаємодію для кожного користувача.

1. Взаємодія з Telegram-ботом

Unity-гра ініціалізується в контексті Telegram-бота. При запуску користувач переходить у WebApp або запускає гру з окремого застосунка, де:

- бот надсилає у Unity-ігровий клієнт спеціальний токен або об'єкт, що містить telegram_id та username;
- ці дані є унікальними і використовуються для ідентифікації користувача в системі;
- якщо користувач запускає гру вперше, Unity надсилає запит на створення облікового запису;
- при наступних запусках відбувається автентифікація на основі Telegram ID.

2. Використання корутин у Unity

Unity використовує асинхронні корутини для обміну даними з сервером через HTTP-запити. Це дозволяє:

- не блокувати головний потік гри під час очікування відповіді;
- гнучко реалізувати логіку типу: *"надіслати запит → дочекатися відповіді → оновити інтерфейс або зберегти стан"*

3. Інтеграція API-сервера

Unity-гра комунікує з сервером за допомогою **HTTP-запитів з JSON-даними**:

- кожна дія (наприклад, клік, покупка покращення, зміна налаштувань, оновлення лідерборду) супроводжується відправленням відповідного запиту;
- дані у відповідь (наприклад, топ-5 гравців, новини) надсилаються у форматі JSON, який Unity парсить і відображає у відповідних UI-панелях;
- всі запити включають Telegram ID для правильної ідентифікації користувача.

4. Синхронізація прогресу

Ключовим завданням інтеграції є **синхронізація ігрового прогресу** між клієнтом і сервером:

- при кожному суттєвому оновленні (зміна рахунку, купівля покращення) Unity надсилає дані на сервер;
- після запуску гри сервер надсилає клієнту актуальний стан користувача, що дозволяє продовжити гру з останнього місця.

5. Обробка новин

Unity також отримує **список новин** через окремий запит до API. JSON-документ з новинами формується іншим Telegram-ботом і зберігається в базі даних. При запуску новинної панелі:

- Unity надсилає запит на /news/latest;
- у відповідь отримує список новин;
- дані парсяться і відображаються в інтерфейсі користувача.

2.3.3 Тестування та відлагодження застосунку

Процес тестування та відлагодження (налагодження) застосунку є важливою частиною життєвого циклу розробки. Він забезпечує стабільну роботу всіх компонентів, виявлення помилок на ранніх етапах і гарантує збереження та цілісність ігрового прогресу користувача.

1. Рівні тестування

У проєкті використовувались кілька рівнів тестування:

- **Модульне тестування (Unit Testing)**

Застосовувалося переважно на стороні Java API-сервера. Було протестовано:

- логіку збереження/отримання ігрових даних;
- валідацію вхідних параметрів;
- відповідність JSON-структур. Json.NET (Newtonsoft) підтримує LINQ-to-JSON, потокову обробку та кастомні конвертери для серіалізації складних об’єктів [15].

- **Інтеграційне тестування**

Перевірялась коректність взаємодії між Unity-грою, API-сервером та базою даних. Зокрема:

- надсилання і обробка HTTP-запитів із Unity;
- збереження прогресу гравця в БД;
- отримання та відображення новин і лідерборду.

- **Системне тестування**

Виконувалося для всієї системи в цілому. Основна увага приділялась:

- сценаріям першого входу гравця;
- перевірці оновлення інтерфейсу після збереження/отримання даних;
- переключенню між панелями (магазин, лідерборд, налаштування).

- **Ручне тестування (Manual QA)**

Було проведено кілька сесій тестування за участі тестувальників або інших користувачів, які виконували основні дії:

- кліки;

- покупки в магазині;
- зміну налаштувань;
- перевірку лідерборду після збереження.

2. Засоби відлагодження

Для виявлення і усунення помилок використовувалися такі інструменти:

- **Unity Console**

Використовувалася для логування дій гравця, помилок запитів, відповідей сервера та внутрішніх винятків.

- **Postman**

Активно використовувався для тестування REST API-сервера. За його допомогою:

- перевірялися відповіді на GET/POST/PUT-запити;
- валідувались JSON-структури;
- тестувались помилкові сценарії (відсутній параметр, помилковий тип тощо).

- **Spring Boot Actuator + Logs**

Сервер логував усі запити та винятки. У випадку помилок на боці сервера можна було швидко виявити, що саме не так з даними, які надсилає Unity.

- **pgAdmin + SQL-інспекція**

Використовувались для перегляду й перевірки даних у базі даних PostgreSQL. Це дозволяло переконатись, що збереження/зчитування даних відбувається правильно. PostgreSQL 14 гарантує ACID-транзакції, відповідність стандарту SQL і розширюваність через додаткові типи й функції [13].

3. Типові помилки та способи їх усунення

У ході тестування були виявлені та усунені типові проблеми:

- неправильне формування JSON у Unity → вирішено шляхом використання JsonUtility або Newtonsoft.Json. JsonUtility у Unity

забезпечує швидку серіалізацію та десеріалізацію об’єктів у форматі JSON для збереження стану гри та обміну з сервером [3];

- занадто довгі відповіді сервера → додано стиснення або оптимізацію запитів;
- проблеми з асинхронним оновленням UI → впроваджено коректну обробку у StartCoroutine.

4. Автоматизоване тестування (опціонально)

У перспективі можливе впровадження **автоматизованого UI-тестування** Unity-гри за допомогою Unity Test Framework або зовнішніх бібліотек, таких як Appium (для мобільної версії).

Розділ 3. РЕЗУЛЬТАТИ РОЗРОБКИ ЗАСТОСУНКУ

Результатом реалізації дипломного проєкту «happy_dog_bot» є єдина інтегрована система, яка забезпечує гейміфіковану взаємодію користувачів через Telegram і складається з клієнтської частини на Unity та серверного бекенду на Java Spring з реляційною базою даних Supabase. Після авторизації через Telegram WebApp API користувач відразу потрапляє на основний ігровий екран (Game Panel), де кожен натиск на зображення собаки відправляє запит на ендпоінт /api/game-state, що зчитує та оновлює поточний рахунок у реальному часі; інтерактивна реакція Unity та асинхронний обмін даними через WebSocket гарантують миттєве відображення приросту очок без перезавантаження.

Вбудований магазин (Shop Panel) дозволяє купувати бонуси — додаткові кліки, пасивний дохід і тимчасові бустери — через REST-запит /api/purchases із механізмом транзакційного відкату у разі помилки, а інформація про придбані поліпшення зберігається у таблиці purchases із подальшим відображенням у грі.

Панель новин (News Panel) автоматично підтягує останні повідомлення з Telegram-каналу бота через /api/news, формує динамічний список анонсів і забезпечує відкриття повного тексту повідомлення в модальному вікні, що дозволяє користувачеві швидко ознайомитися з важливими оновленнями. Рейтинг гравців (Leaderboard Panel) формується на основі зібраних даних із сесій та покупок, сортується за параметрами «за добу» та «за тиждень» і кешується для прискорення завантаження, забезпечуючи відображення топ-10 гравців без затримок.

Панель налаштувань (Settings Panel) дає змогу користувачам змінювати мову інтерфейсу, регулювати гучність звукових ефектів та вибирати темну або світлу тему, причому будь-які правки миттєво зберігаються через /api/settings і застосовуються одразу завдяки real-time підключенню.

Серверна частина реалізована за мікросервісною архітектурою з REST-контролерами для обробки запитів користувачів, сервісами бізнес-логіки для валідації та обробки транзакцій, шаром доступу до даних (DAO) для взаємодії з Supabase, а також вбудованим моніторингом через Spring Actuator і візуалізацією метрик у Grafana.

Для деплою використано Docker-контейнери на платформі Railway з налаштованим CI/CD — автоматичним запуском unit- і integration-тестів, виконанням міграцій бази даних та безперервним оновленням сервісів без простоїв. Завдяки такій інфраструктурі проект «happy_dog_bot» демонструє високу продуктивність, надійність і масштабованість, дозволяє оперативно випускати нові функції та гарантує ефективне адміністрування і поглиблену аналітику через стандартизований Swagger-документований API.

В результаті роботи над проектом були створені наступні сторінки:

- Головна сторінка гри
- Панель магазину
- Панель таблиці лідерів
- Панель новин
- Панель налаштувань

Головна сторінка (рис. 3.1) – це сторінка, на якій знаходяться ключові ігрові елементи.

На головному екрані аплікації «happy_dog_bot» у Telegram WebApp користувач бачить центральну круглу кнопку з зображенням собаки, котру необхідно клікати для накопичення балів: при кожному натисканні відтворюється характерний гавкіт, а сама кнопка анімовано стискається при натисканні й плавно повертається до початкового розміру після відпускання, у той час як баланс у вигляді великого цифрового індикатора над кнопкою (наприклад, 881$) оновлюється у реальному часі.

Позаду основного елементу інтерфейсу постійно анімовано падаючі монети, що створює відчуття безперервного заробітку.

Рисунок 3.1 Головна сторінка

Під кнопкою розташовано три основні навігаційні панелі: панель новин (іконка газети), панель таблиці лідерів (іконка трофея) та панель магазину (іконка будиночка), при наведенні курсору або торканні кожна кнопка отримує «hover»-анімацію зі збільшенням масштабу та плавним ефектом підсвічування, що підвищує зручність інтерфейсу. У верхній частині екрана міститься панель налаштувань (іконка шестерні), через яку можна змінювати мову інтерфейсу, регулювати гучність звуку та перемикати тему оформлення. Весь інтерфейс виконано в єдиному стилі Unity WebGL із яскравою синьо-золотистою палітрою, чіткими іконками та інтерактивними анімаціями, що забезпечують інтуїтивно зрозумілий та приємний досвід взаємодії.

Панель магазину (рис. 3.2) – панель, в якій можна купити внутрішньоігрові апгрейди.

На екрані магазину (Shop Panel) відображається заголовок «Shop» у верхній частині та кнопка закриття (X) для повернення на головний екран. Нижче розміщені два великих елементи Button для придбання апгрейдів: «Click Booster 2000$» та «Passive Income 2000$» — при натисканні на кожну з кнопок відтворюється характерний дзвін монет, а вартість покупки збільшуеться в два рази. Успішна покупка підтверджується миттєвим оновленням балансу. Нижня частина інтерфейсу зберігає навігаційну панель із іконками News, Leaderboard і Shop для швидкого перемикання між розділами, а візуальні стилі, анімації та палітра відповідають загальному дизайну застосунку.

Панель таблиці лідерів(рис. 3.3) – це панель, в якій відображаються гравці з найбільшою кількістю очок Score.

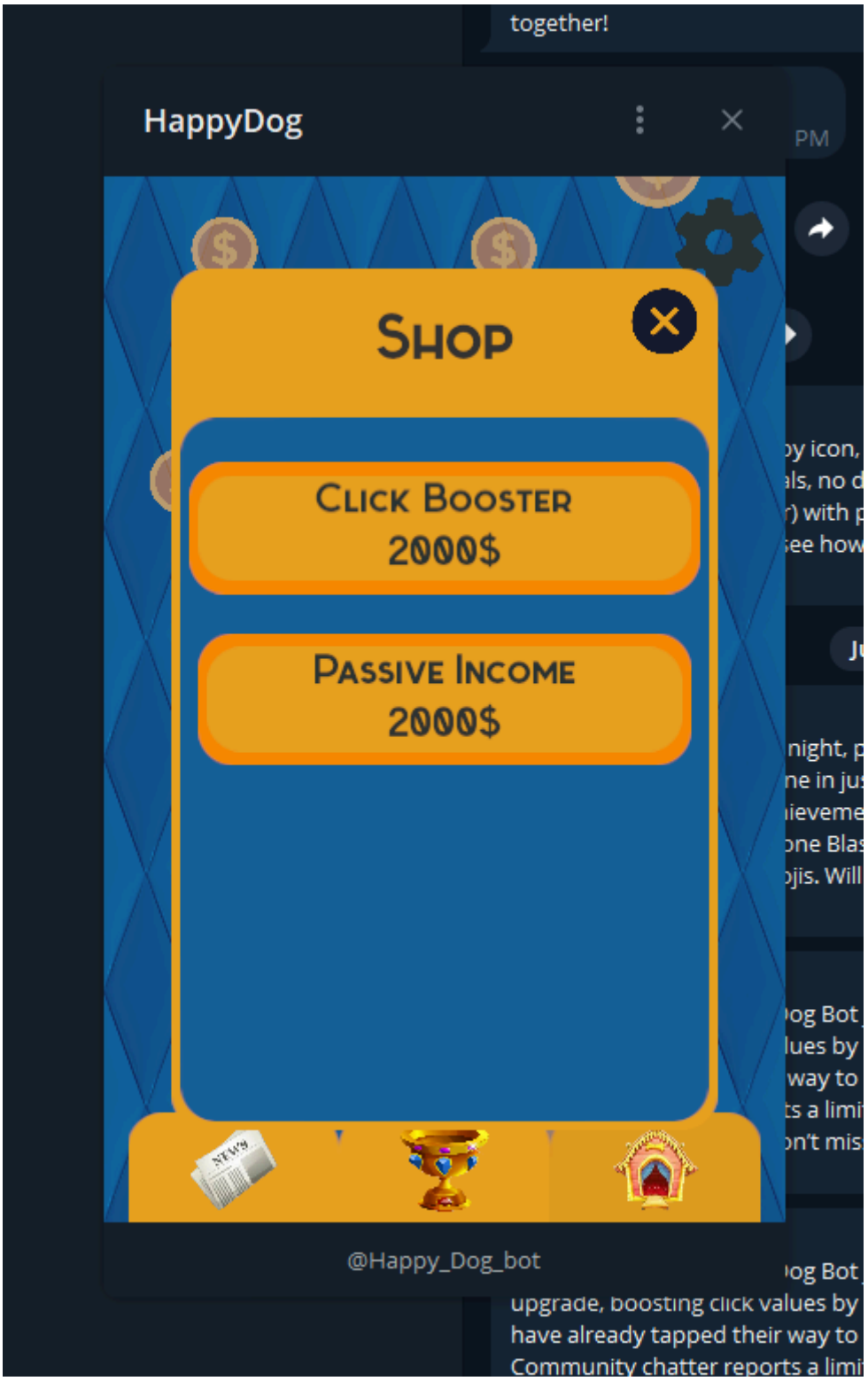

Рисунок 3.2 – Панель магазину

На панелі таблиці лідерів (Leaderboard Panel) у верхній частині розміщено заголовок «Leaderboard» та кнопку закриття (X) для виходу до головного екрана; над списком бачимо декоративне зображення трьох трофеїв

(золотий, срібний, бронзовий), яке підкреслює ранжування. Нижче розташований список топ-5 гравців, де кожен рядок складається з номера місця у колі, нікнейму та кількості очок (наприклад, 1 Vladsleyvv 11512); перше місце підсвічується жовтою рамкою, друге — срібною, третє — бронзовою, а інші — у нейтральному стилі. Панель підтримує реальне оновлення даних через WebSocket, тож зміни в рейтингу відображаються миттєво. Внизу зберігається навігаційна панель із іконками News, Shop і Leaderboard для швидкого переходу між розділами, а вся панель оформлена в єдиному стилі з яскравою синьо-золотистою палітрою та інтерактивними анімаціями, які забезпечують цілісний UX.

Панель таблиці лідерів (рис. 3.4) – це панель, в якій відображаються новини, що надходять з телеграм каналу.

На панелі новин (News Panel) заголовок «News» на жовтому фоні та кнопка закриття (X) у правому куті дозволяють повернутися до головного екрана. Під заголовком розташовано ScrollView із вікном-viewport, у якому вертикально прокручуються жовті блоки з аватаркою бота, назвою «HappyDog» та тілом повідомлення. Кожен блок динамічно заповнюється після того, як модуль NewsManager через REST-запит до API Telegram-бота отримує свіжі пости з каналу (рис. 3.5), тому вміст панелі точно відповідає тому, що видно в чаті. У футері панелі знаходяться іконки переходу між розділами (газета – News, кубок – Leaderboard, будка – Shop). Весь інтерфейс оформлено в єдиному яскраво-синьо-жовтому стилі з плавними анімаціями, що забезпечує цілісний і зручний UX.

Рисунок 3.3 – Панель таблиці лідерів

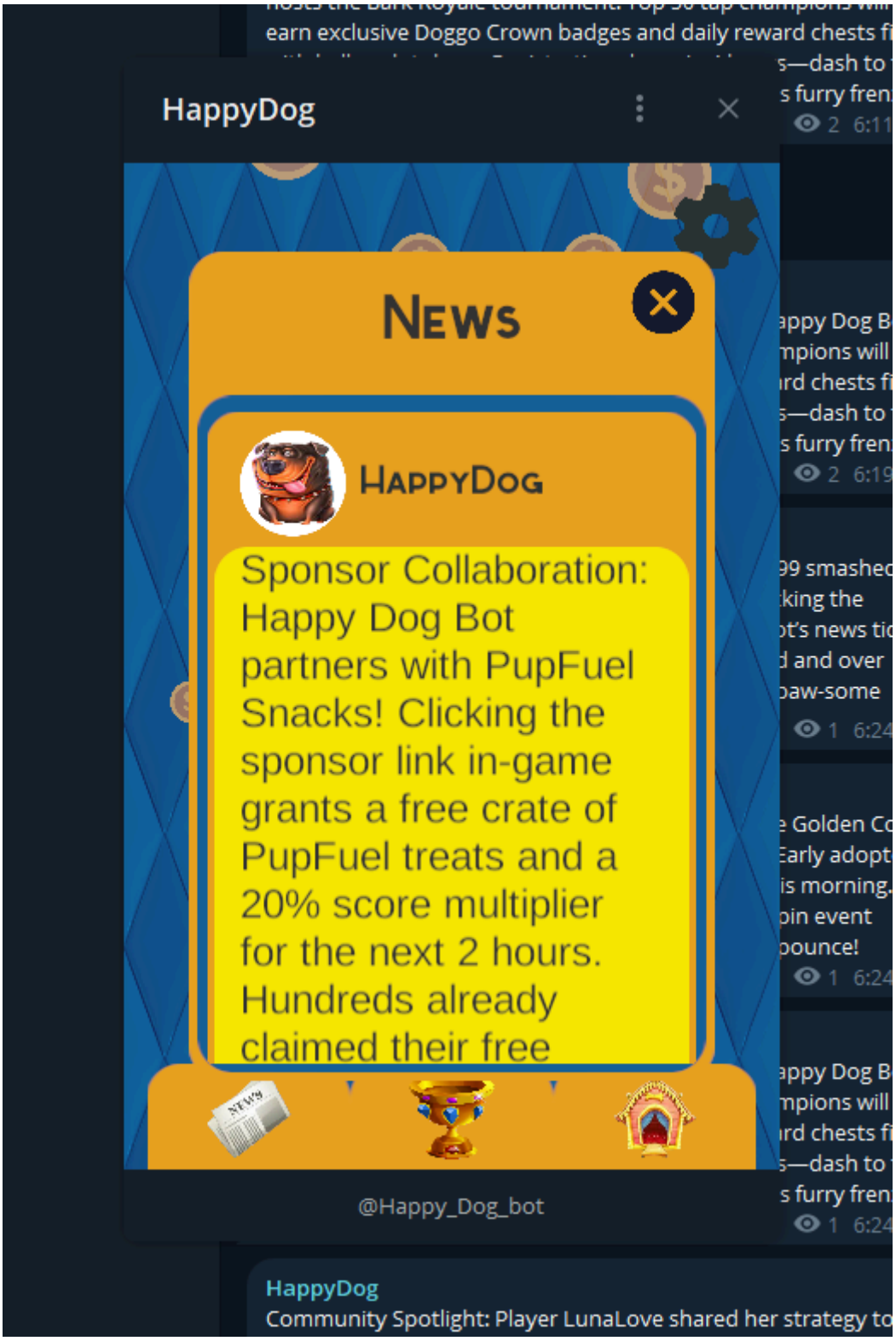

Рисунок 3.4 – Панель таблиці новин

Рисунок 3.5 – Новини з телеграм каналу

Панель новин (рис. 3.6) – це панель, в якій користувач може змінювати налаштування для свого клієнту гри.

На панелі налаштувань (Settings Panel) зверху розташовано заголовок «Settings» та кнопку закриття (X) для повернення до головного екрана, під якими знаходяться дві великі круглі кнопки: іконка динаміка для регулювання гучності звукових ефектів та іконка ноти зі забороною для відключення музики. При натисканні на кнопки відтворюється анімація зміни спрайту; будь-які зміни застосовуються миттєво через REST-запит та зберігаються в базі даних.

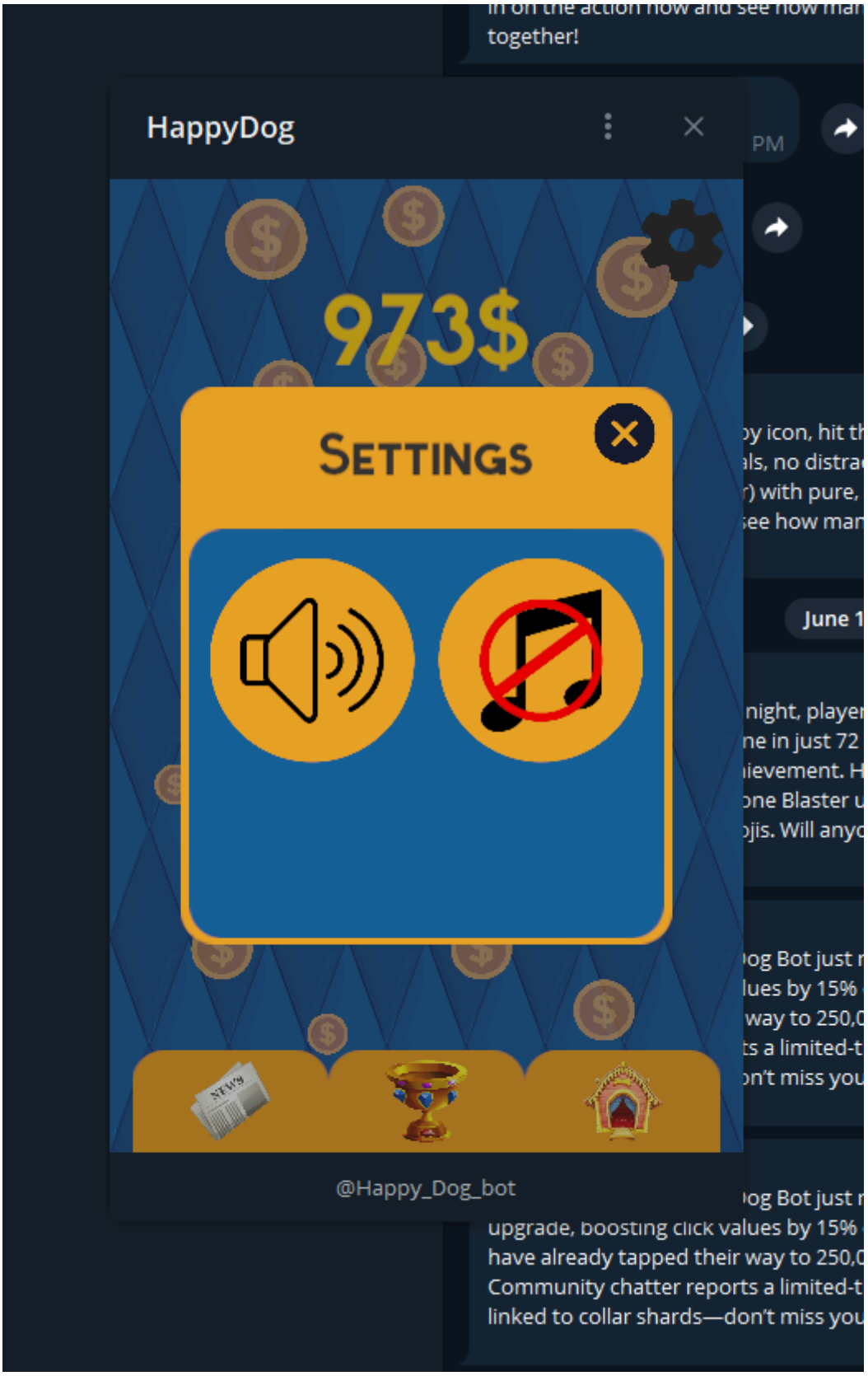

Рисунок 3.6 – Панель новин

Розділ 4. ЕРГОНОМІКА І ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

Цей розділ має на меті всебічно проаналізувати доцільність створення програмного продукту як з точки зору зручності використання, так і з позицій витрат і очікуваної ефективності.

У першій частині — **ергономічній** — основна увага зосереджується на оцінці інтерфейсу та зручності взаємодії користувача з застосунком. Гра типу "клікер" повинна бути інтуїтивно зрозумілою, візуально привабливою та простою у навігації. В процесі розробки я врахував принципи UX-дизайну: доступність основних функцій у 1–2 натисканнях, баланс між візуальною простотою та ігровою привабливістю, а також гнучкість налаштувань (звук, частота оновлень, мова інтерфейсу). Застосунок має мінімалістичний дизайн з елементами, стилізованими під казино-естетику, що підсилює гейміфікаційний ефект та підтримує мотивацію гравця.

У другій частині розглядається **техніко-економічне обґрунтування**. Вона охоплює такі аспекти, як обсяг витрачених ресурсів, використані технології та їхня вартість, а також оцінку потенційних вигод від розробки. Оскільки проєкт реалізовано з використанням відкритого рушія Unity, безплатного хостингу Railway та вільно доступної СУБД MySQL, грошові витрати були мінімальними. Основним ресурсом стала особиста праця над ідеєю, програмуванням, тестуванням та інтеграцією з Telegram API. Водночас, у разі подальшої монетизації, застосунок може забезпечити дохід через рекламу, внутрішні покупки або донати.

З технічної точки зору, продукт масштабований, з можливістю додавання нових функцій (наприклад, нових типів бонусів, PvP-режиму тощо) та перенесення на інші платформи (ПК, iOS). Архітектура проєкту передбачає гнучкість для подальшого оновлення без кардинальних змін коду.

4.1 Вимоги до програмного забезпечення з погляду користувача

З позиції кінцевого користувача програмне забезпечення у вигляді клікер-гри має відповідати низці важливих вимог, які визначають рівень задоволеності, зручність використання та бажання продовжувати взаємодію з застосунком. Основні вимоги такі:

- **Інтуїтивність інтерфейсу.** Усі основні дії — натискання на головного персонажа (собаку), перегляд новин, перехід у магазин чи до налаштувань — мають бути максимально простими у доступі. Користувач повинен інтуїтивно розуміти, як виконати певну дію без попереднього ознайомлення з інструкцією.
- **Висока швидкодія.** Гра має моментально реагувати на дії користувача, особливо під час кліків. Затримки або збої у відгуку можуть спричинити втрату інтересу до застосунку.
- **Адаптивність дизайну.** Інтерфейс повинен бути зручним на різних розмірах екранів мобільних пристроїв. Усі кнопки мають бути достатнього розміру, щоб ними було легко користуватися навіть на невеликих екранах.
- **Відчуття прогресу.** Гравець очікує, що з кожною взаємодією він просувається вперед: отримує монети, відкриває нові можливості, бачить результат своїх дій. Це підвищує мотивацію до продовження гри.
- **Наявність налаштувань.** Можливість увімкнути або вимкнути звук, змінити мову або переглянути оновлення гри дає користувачеві відчуття контролю.
- **Інформаційна прозорість.** Гравець має легко дізнаватися скільки він заробив монет, які бонуси активні, скільки коштує те чи інше покращення, а також мати змогу переглядати новини чи повідомлення, що надходять із Telegram.
- **Збереження прогресу.** Гравець очікує, що навіть після закриття гри його досягнення будуть збережені, і він зможе продовжити з того місця, на якому зупинився.

- **Інтеграція з Telegram.** Важливою вимогою з боку користувача є зручний доступ до новин і сповіщень через Telegram. Такий підхід дозволяє отримувати оновлення в реальному часі, що підвищує рівень залученості, а також створює додаткову точку взаємодії із застосунком без потреби постійно заходити в гру.

4.2. Ергономічні цілі і показники якості продукту

У процесі розробки програмного забезпечення особливу увагу було приділено ергономіці. Йдеться не лише про візуальний комфорт користувача, а й про загальний досвід взаємодії з інтерфейсом гри, його інтуїтивність, передбачуваність, швидкість реакції та приємність візуального сприйняття.

Одним із ключових завдань стало зменшення когнітивного навантаження. Інтерфейс реалізовано таким чином, щоб новий користувач міг розібратися в основних можливостях гри буквально за кілька секунд. Навігація між основними розділами – кліком по головному персонажу, переходом у магазин, переглядом новин чи відкриттям налаштувань – не потребує жодних додаткових інструкцій. Усі елементи розташовані логічно, а їхні розміри й контрастність забезпечують зручне керування навіть на невеликих екранах мобільних пристроїв.

Не менш важливою була і візуальна складова. Стилістику гри розроблено з урахуванням сучасних тенденцій мобільного геймінгу: фон виконано у спокійних синіх відтінках, графіка мінімалістична, без зайвих візуальних подразників, але водночас не позбавлена характеру. Образ собаки – американського ротвейлера – виступає центральним акцентом, привертаючи увагу й викликаючи емоційний зв’язок із користувачем.

Оскільки гра розрахована на короткі ігрові сесії, важливо було зробити так, щоб кожен вхід мав значення. Навіть кілька хвилин у грі дають відчуття прогресу, що стимулює бажання повертатись знову. Для цього реалізовано як активну взаємодію (кліки), так і пасивний приріст монет, що дозволяє отримувати результат без постійного перебування в додатку.

Показники якості гри оцінюються через декілька параметрів. Насамперед, це легкість освоєння – середньостатистичному користувачеві достатньо менше хвилини, щоб зрозуміти, як функціонує ігрова механіка. Окрім цього, важливим індикатором виступає точність введення – тобто наскільки просто влучати у потрібні елементи інтерфейсу, особливо на сенсорних екранах. За результатами тестування більшість користувачів не мали труднощів у взаємодії з грою. Позитивно оцінено також загальну читабельність – шрифти підібрані таким чином, щоб інформація була добре помітною навіть при денному освітленні або зменшеному яскравому режимі екрана.

Окрім загальних ергономічних принципів, доцільно виокремити також низку кількісних показників, які дозволяють об’єктивно оцінити якість користувацького досвіду. Для цього було сформовано перелік цільових метрик, що мають значення у контексті мобільної клікер-гри:

1. Середній час освоєння основної механіки (Time to Learn Core Mechanics). Мета — не більше 20 секунд. Цей показник вимірюється під час тестування новими користувачами: фіксується момент першого відкриття гри до першої цільової дії — натискання на собаку, відкриття магазину або перегляду новин.

2. Рівень точності натискань (Touch Precision Rate). Цільове значення — 95% і вище. Показник розраховується як співвідношення успішних натискань на інтерактивні елементи до загальної кількості натискань. Якщо користувач часто "промахується" — це свідчить про ергономічні недоліки, зокрема у розмірах елементів або їх розміщенні.

3. Середній час виконання типових дій (Avg. Task Completion Time). Наприклад, відкриття магазину має займати не більше 2 секунд після запуску гри. Подібний підхід дозволяє визначити, наскільки інтерфейс підтримує швидку і комфортну навігацію.

4. Кількість помилкових дій (Error Rate). Оптимальне значення — не більше 1 помилки на 100 дій. До помилкових дій відносяться натискання не на ті кнопки, що очікувались, закриття меню без виконання цільової дії тощо.

5. Рівень задоволеності інтерфейсом (Satisfaction Score). Оцінюється через анкетування користувачів за шкалою від 1 до 5. Ціль — середнє значення не менше 4. Це суб'єктивна метрика, однак вона дозволяє отримати якісний зворотний зв'язок щодо дизайну та зручності гри.

4.3 Результат реалізації ергономічних цілей

Для попередньої перевірки ергономічної якості реалізованого інтерфейсу було проведено обмежене тестування з участю 12 користувачів віком від 16 до 28 років. Усі учасники не були ознайомлені з проєктом раніше, що дозволило змоделювати досвід "нового гравця". Тестування проходило на мобільних пристроях із різними розмірами екранів (від 5.8 до 6.7 дюймів), а дані фіксувались вручну та за допомогою вбудованої аналітики.

Результати оцінювання таких ключових показників виявились наступними:

1. Середній час освоєння основної механіки (Time to Learn Core Mechanics). У середньому користувачі здійснювали перший осмислений клік по собаці вже через **6 секунд** після запуску гри. Це свідчить про інтуїтивність інтерфейсу, відсутність потреби у додаткових поясненнях і відповідність меті (не більше 20 секунд).

2. Рівень точності натискань (Touch Precision Rate). Із загальної кількості зафіксованих натискань (понад 1300) — **96,2%** виявились точними. Основні помилки були пов'язані з малим екраном на одному з пристроїв і стосувались зони переходу до магазину. Висновок — переважна більшість елементів розташовані та масштабовані коректно.

3. Середній час виконання типових дій (Avg. Task Completion Time)

- Відкриття магазину: **1,3 с**
- Перехід до вкладки "Новини": **1,8 с**

- Відключення музики: **2,0 с**

Усі ці значення перебувають у межах встановленого порогу у 2 секунди, що підтверджує ефективність навігаційних рішень.

4. Кількість помилкових дій (Error Rate). Було зафіксовано **8 помилкових натискань** із загальної кількості **986 дій**, що становить **0,81%**. Це відповідає встановленій нормі (менше 1%) і не потребує критичних змін, хоча декілька учасників звернули увагу на дещо не інтуїтивне зображення піктограми списку кращих гравців.

5. Рівень задоволеності інтерфейсом (Satisfaction Score). Суб’єктивна оцінка зручності та візуального оформлення, зібрана через Google-форму після тесту, дала такі результати:

- Середнє значення: **4,3 / 5**
- Найчастіше згадувані позитивні риси: "простота", "приємний дизайн", "інтуїтивність"
- Основні побажання: "додати анімації", "урізноманітнити фон"

Отримані результати дозволяють стверджувати, що реалізовані ергономічні цілі досягнуті в межах первинного MVP-рівня гри. Подальша робота має фокусуватись на адаптації інтерфейсу під найменші екрани, а також розширенні функціональності без шкоди зручності користування.

Всі отримані дані представлені нижче в таблиці 4.1.

Таблиця 4.1 – Результати ергономічного дослідження

№ учасника	Час до першого кліку (с)	Точність натискань (%)	Середній час дії (с)	К-сть помилок (%)	Задоволення (1–5)
1	8	96,2	1,40	0,9	5
2	5	97,1	1,20	0,7	4
3	3	95,5	1,70	1,1	4
4	3	96,9	1,30	0,6	5
5	11	94,8	1,60	0,8	4
6	5	96,7	1,30	0,5	5
7	9	96,3	1,50	1,0	4
8	2	95,9	1,40	0,9	4
9	7	96,1	1,80	1,2	3
10	8	97,4	1,10	0,4	5
11	5	95,3	1,60	1,0	4
12	6	96,5	1,30	0,7	5
Середнє Значенн	6	96,225	1,433333	0,816667	4,333333333

4.4 Техніко-економічне обґрунтування розробки

Розробка програмного забезпечення, зокрема мобільного додатку у вигляді клікер-гри, повинна не лише відповідати технічним та ергономічним критеріям, але й бути виправданою з економічної точки зору. У межах цього розділу здійснено аналіз витрат на створення застосунку, оцінено потенційну рентабельність, а також визначено доцільність реалізації проєкту в сучасних умовах.

Проєкт реалізовувався переважно за допомогою безкоштовних або умовно безкоштовних інструментів. Unity використовувався як основна платформа для розробки — завдяки безкоштовній Community-версії, не виникало додаткових ліцензійних витрат. Для зберігання та обробки даних було обрано хмарний сервіс Railway, який надає безкоштовні ресурси в межах базових тарифів. Завдяки цьому вдалося мінімізувати початкові витрати без шкоди для функціоналу.

Телеграм-канал, звідки імпортуються новини, також не потребує фінансових витрат, адже інтеграція з API Telegram дозволяє автоматизовано оновлювати базу даних. Таким чином, підтримка контенту реалізована без додаткових витрат на хостинг чи редагування вручну.

Основна стаття витрат — це час розробника. Всі інші складові — публікація гри, використання безкоштовного хостингу на перших етапах, просування через Telegram — не потребували значних фінансових ресурсів.
Водночас, уже на етапі проєктування було передбачено чітку стратегію **майбутньої монетизації**. Основними джерелами прибутку стануть:

- **Інтеграція внутрішньої реклами** через рекламні мережі, такі як Google AdMob або Unity Ads. Формат банерів або відеореклами планується адаптувати до ігрового процесу, зокрема для отримання бонусів чи пришвидшення прогресу гравця.
- **Залучення рекламодавців через систему завдань.** Гравцю пропонуватимуть прості дії, як-от підписка на певний Telegram-канал або перехід на сторонній сайт. За виконання таких дій користувач отримуватиме внутрішню валюту або інші заохочення, а рекламодавець — живу аудиторію. Це дозволяє створити альтернативну модель монетизації без необхідності прямих фінансових витрат від гравця.

Очікувана ефективність такого підходу зростає при нарощуванні користувацької бази. Наприклад, при середньому доході в **5 грн з одного активного гравця на місяць**, дохідність на рівні **20 000 грн/міс** можлива вже при **4 000 активних користувачах**. Враховуючи невисоку собівартість проєкту, точка беззбитковості досягається досить швидко, а подальше масштабування відкриває можливості для стабільного прибутку або реінвестування.

4.4.1 Резюме продукту

У межах дипломної роботи було реалізовано програмний продукт у вигляді мобільної клікер-гри «Dog Clicker», створеної з використанням рушія Unity та інтегрованої з базою даних MySQL, розміщеною на хмарному сервісі Railway. Головна ідея гри полягає у натисканні на зображення собаки (порода — американський ротвейлер), за що користувач отримує внутрішню валюту,

яку згодом можна витратити на покращення, що підвищують ефективність гри.

Проєкт передбачає простий, інтуїтивно зрозумілий інтерфейс, що адаптований до мобільних пристроїв та підтримує українську мову. Додатковими функціональними модулями є магазин із покращеннями, вкладка з новинами, які автоматично надходять із Telegram-каналу, та сторінка налаштувань, де можна регулювати звук та інші параметри.

З технічного боку, застосунок демонструє взаємодію клієнтської частини, створеної в Unity, із серверною базою даних, що зберігає інформацію про користувачів, прогрес гри, новини та інші ключові об’єкти. Такий підхід забезпечує масштабованість проєкту й дозволяє гнучко керувати контентом.

Особливу увагу в реалізації було приділено зручності доступу. Гравець має змогу почати гру без реєстрації, зберігати прогрес на сервері та отримувати новий контент через Telegram у реальному часі. Передбачено також потенційні канали монетизації: внутрішню рекламу та інтеграцію завдань від рекламодавців.

4.4.2 Опис розробленого продукту

Проєктований продукт — це інтерактивна Telegram-гра під назвою **happy_dog_bot**, що реалізована на рушії Unity з повною інтеграцією backend-сервера, побудованого на Java Spring Boot, та бази даних Supabase. Основна ідея гри полягає у клік-механіці: користувач, натискаючи на стилізовану кнопку з собакою, заробляє внутрішньоігрові очки (score), які зберігаються та обробляються через зовнішній сервер. Гра є частиною Telegram-бота і не потребує окремого встановлення, що робить її максимально доступною для цільової аудиторії.

Happy_dog_bot має яскраво оформлений інтерфейс, у якому реалізовано декілька ключових ігрових екранів: головне меню з кнопкою для взаємодії, панель магазину, що дозволяє придбати покращення для активного та

пасивного заробітку очок, панель лідерборду, яка демонструє топ-5 гравців за кількістю набраного score, а також модуль новин, де користувач бачить повідомлення з Telegram-каналу. Крім того, існує панель налаштувань, у якій можна вимкнути або ввімкнути музику та звукові ефекти — ці параметри також зберігаються в базі даних та автоматично завантажуються при наступному вході в гру.

Вся логіка взаємодії між грою та сервером реалізована за допомогою корутин у Unity. Через HTTP-запити здійснюється надсилання даних до backend-сервера, де розгорнуто REST API. Цей сервер обробляє запити, пов’язані зі створенням та оновленням користувача за Telegram ID, зчитуванням даних для лідерборду, а також з додаванням та оновленням новин. Реєстрація користувача в системі відбувається автоматично під час першого входу, при цьому Telegram ID та Username записуються до відповідної таблиці бази даних. Крім основного score, зберігаються також параметри пасивного та активного доходу (PassiveScore, ClickScore), які впливають на швидкість накопичення очок у грі.

Інформація про користувача зберігається у таблиці Supabase, яка пов’язана також із таблицею налаштувань — тут зберігається стан параметрів звуку та музики. Друга важлива частина бази — це таблиця новин, що отримує контент із Telegram-каналу. Для цього використовується окремий бот, реалізований у середовищі BotFuzzer, який зчитує повідомлення з каналу, передає їх у вигляді рядкових запитів до backend-сервера, після чого вони зберігаються в базі даних і з’являються в інтерфейсі гри. Таким чином, система дозволяє не лише інтерактивно грати, але й отримувати актуальні повідомлення безпосередньо в ігровому середовищі.

Результатом розробки стало сучасне рішення, яке поєднує Telegram, Unity, Java та Supabase в одному цілісному продукті — **happy_dog_bot**, що демонструє глибоку інтеграцію різних технологій та може слугувати прикладом ефективного поєднання гейміфікації й мобільного

мікросервісного підходу. Supabase надає альтернативу Firebase із миттєвими REST-API, real-time підписками та аутентифікацією поверх PostgreSQL [12].

4.4.3 Оцінка ринку збуту

У цьому підрозділі здійснюється глибокий аналіз ринкових перспектив інтерактивного програмного продукту **happy_dog_bot** — кросплатформеної Telegram-гри з клікерною механікою, яка реалізована за допомогою рушія Unity та має інтеграцію з backend-сервером, побудованим на Java Spring Boot, а також базою даних Supabase. Така архітектура дозволяє забезпечити не лише стабільну роботу гри, а й ефективну обробку користувацьких даних у реальному часі. Основна ціль полягає в створенні зручного, швидкодоступного цифрового продукту, який поєднує гейміфікацію, розваги та інтеграцію з Telegram-екосистемою, що наразі активно розвивається як платформа для міні-додатків.

Ігровий продукт спрямований на широку вікову аудиторію, починаючи з молодшої школи (10+ років) і до віку 30 років, що дозволяє ефективно працювати з кількома демографічними категоріями. Враховуючи механіку гри, простоту інтерфейсу, а також відсутність потреби в додатковому встановленні чи високопродуктивних пристроях, **happy_dog_bot** орієнтований на користувачів, які бажають проводити дозвілля у Telegram, поєднуючи розваги з елементами досягнень та соціального змагання. Відповідний аналіз цільової аудиторії подано в таблиці 4.1, яка відображає поведінкові та соціальні характеристики різних сегментів користувачів.

Таблиця 4.1 — Аналіз цільової аудиторії

Категорія користувачів	Опис
Школярі та підлітки	Користувачі віком 10–17 років, які шукають прості розваги у зручному форматі. Для них важлива яскрава графіка, швидка взаємодія та прості правила гри.
Студенти	Користувачі віком 18–24 років, здебільшого активні в Telegram, зацікавлені у казуальних іграх у перервах між навчанням. Цінують

	можливість змагатися з друзями та прогресувати.
Молоді спеціалісти	Особи віком 22–30 років, які працюють або тільки починають професійну діяльність. Вони шукають легкий спосіб розвантаження, особливо у форматі, який не потребує інсталяції.
Казуальні гравці	Люди, які грають нечасто, але залучаються завдяки простому геймплею та Telegram-доступу. Можуть стати частиною гри завдяки лідерборду або системі досягнень.
Геймери та ентузіасти Telegram-ботів	Досвідчені користувачі Telegram-ботів, зацікавлені в нових ігрових механіках і інтеграціях. Готові брати участь у тестуванні, змаганнях, івентах.

Доповнюючи ці сегменти, варто зазначити, що серед гравців спостерігається зростаюча потреба в легких, швидкодоступних, але водночас стильних іграх із мінімальним когнітивним порогом. Telegram у цьому контексті виконує роль платформи нового покоління, яка конкурує з App Store і Google Play завдяки своїй універсальності, швидкості розповсюдження контенту та мінімальному порогу входу. Завдяки впровадженню Telegram Web Apps користувачі можуть взаємодіяти з грою напряму з чат-інтерфейсу, що істотно пришвидшує охоплення. Telegram Web Apps дозволяють відображати кастомні веб-сторінки всередині інтерфейсу Telegram з безпечним обміном даними між HTML/JS і ботом [6].

Середній портрет користувача додатково деталізовано в таблиці 4.2. З погляду маркетингового підходу, це дає змогу сформувати більш адресну стратегію просування гри та визначити ключові точки дотику з аудиторією.

Таблиця 4.2 — Середньостатистичний гравець happy_dog_bot

Аспект	Характеристика
Вікова група	10–30 років. В основному — школярі, студенти, молоді працівники.
Стать	Універсальна. Орієнтована як на хлопців, так і на дівчат.

Освітній рівень	Середній або початковий рівень освіти — учні, студенти, випускники.
Цифрова поведінка	Активні в месенджерах, особливо в Telegram. Більшість — користувачі смартфонів.
Професійна діяльність	Учні, студенти, працівники сфери IT, маркетингу, фрилансери.
Основні потреби	Отримання задоволення, азарт, змагання з друзями, покращення результатів.
Мотивація до гри	Розваги в зручному, швидкодоступному форматі без потреби інсталяції.
Мовна орієнтація	Інтерфейс англійською, хоча основна частина гравців — українці. Гра готова до інтернаціоналізації.

З географічної точки зору запуск гри відбувається з Києва, де спостерігається найбільша концентрація цифрово активної молоді та студентства, однак продукт із самого початку розроблявся з прицілом на міжнародну аудиторію. Весь інтерфейс гри реалізовано англійською мовою, що відкриває широкі можливості для експорту продукту, локалізації під інші ринки та створення мультикультурної геймерської спільноти.

Задля кращого розуміння внутрішньої мотивації користувачів, факторів їхнього повернення до гри, а також для оцінки маркетингового потенціалу сегментів, було побудовано адаптовану таблицю попиту (табл.4.3), яка враховує поведінкові патерни, технічні особливості, соціальні вподобання та комунікаційні канали.

Таблиця 4.3 — Поведінка та попит користувачів гри happy_dog_bot

Тип користувача	Що приваблює в грі	Поводження користувача	Потенціал зростання
Гравці молодшого віку (10–13 років)	Веселий візуальний стиль, анімації, швидка реакція на натискання	Грають короткими сесіями, зазвичай кілька хвилин за раз	Високий при просуванні в шкільних групах
Активні Telegram-користувачі	Можливість грати прямо в Telegram без встановлення	Часто переходять із ботів у Telegram,	Дуже високий завдяки Telegram-екосистемі

	сторонніх додатків	пробують нові ігри	
Користувачі з низькою потужністю пристроїв	Гра не потребує багато ресурсів: швидке завантаження, низьке енергоспоживання	Грають на старих телефонах, цінують простоту	Стабільний, якщо зберігати простоту гри
Любителі змагальних ігор	Присутність лідерборду, стимул до досягнень, бажання бути першим	Повертаються до гри щодня, щоб покращити результати	Залежить від активності у спільнотах
Користувачі, які цінують персоналізацію	Налаштування звуку, прогрес, індивідуальні апгрейди	Залишаються довше, якщо бачать прогрес і мають вплив на гру	Зростає при введенні нових налаштувань чи кастомізації

Загалом, продукт **happy_dog_bot** має високу гнучкість адаптації до нових ринкових умов завдяки своїй мінімалістичній структурі, технологічній простоті й можливості до кастомізації. Telegram як канал розповсюдження має високий органічний потенціал завдяки ефекту вірусності, можливості прямого спілкування з аудиторією через боти, та швидкому поширенню через канали і групи.

Отже, можна зробити висновок, що реалізація та просування гри **happy_dog_bot** на ринку є доцільними з точки зору як технічного виконання, так і ринкової відповідності. Враховуючи динаміку Telegram-платформи, її глобальне зростання, а також зростаючий попит на інтегровані ігри у форматі WebApp, проект володіє високим потенціалом масштабування, повторного використання й комерціалізації в майбутньому.

4.4.4 Стратегія маркетингу

У цьому підрозділі детально розглядається маркетингова стратегія для цифрового продукту **happy_dog_bot**, яка має на меті забезпечення широкого охоплення цільової аудиторії, формування позитивного образу гри та стимулювання зростання активної бази користувачів. Розробка ефективної маркетингової стратегії є ключовим елементом успішного просування в умовах цифрового перенасичення та обмеженого часу уваги користувача. Успіх проєкту залежить не лише від технічної якості, а й від того, наскільки швидко ігрова пропозиція буде помічена, випробувана та підтримана спільнотою гравців.

Оскільки **happy_dog_bot** є Telegram-грою з інтегрованим Unity-інтерфейсом, маркетингові дії повинні зосереджуватись на цифрових платформах, де присутні молодіжні спільноти, IT-аудиторія, а також фанати казуальних мобільних ігор. Головною задачею просування є створення стабільного інтересу до гри, підкріпленого соціальним ефектом (вірусність, шерінг у чатах), системою мотивацій (лідерборд, оновлення, новини) та впровадженням елементів персоналізації.

Основні напрямки маркетингової стратегії:

1. **Telegram-маркетинг** — основний акцент робиться на екосистему Telegram. Реклама в популярних Telegram-каналах, використання телеграм-ботів-партнерів, запуск власного Telegram-каналу новин гри та чатів користувачів для формування ком’юніті. Перевага — швидке поширення через чати, інтеграція глибоких посилань (deep links), push-сповіщення.

2. **SMM-кампанії (TikTok, Instagram, Discord)** — створення коротких відеороликів із геймплеєм гри для TikTok та Reels, а також запуск мем-кампаній. У Discord можливо створити міні-сервер із підтримкою, ролями гравців, анонсами івентів та конкурсами. Це сприяє створенню ігрової субкультури навколо гри.

3. **Реферальна система** — додавання в гру механіки «запроси друга», яка дозволить користувачу ділитися посиланням на гру й отримувати

за це бонуси. Це суттєво впливає на виральність продукту без витрат на платну рекламу.

4. **Промо-івенти та хакатони** — участь у тематичних івентах, онлайн-змаганнях і гейм-джемах. Через участь у конкурсах розробників Telegram WebApp є можливість підвищити впізнаваність гри серед професійної спільноти.

5. **Контент-маркетинг** — створення блогу або серії публікацій, які розповідають про механіку гри, цікаві апдейти, історію створення та філософію «собаки-клікеру». Це також сприяє пошуковій оптимізації в Google.

6. **Рекламні кампанії (PPC)** — запуск платної реклами в Google Ads (з фокусом на WebApp/Telegram-геймерів) та Instagram/TikTok Ads. Кампанії повинні бути орієнтовані на конверсію та виральну залученість.

7. **Аналітика та оптимізація через Supabase і Telegram Bot API** — відстеження поведінки користувачів (retention, session length, churn) через Supabase та використання Telegram Bot API для A/B тестування повідомлень, форматів залучення, привітань тощо.

Таблиця 4.4 — Цілі та інструменти маркетингової стратегії happy_dog_bot

Ціль	Маркетинговий інструмент	Очікуваний ефект
Розширення аудиторії	Telegram-реклама, реферальна програма	Вибухове зростання інсталяцій через шерінг
Залучення молоді 10–25 років	TikTok, Instagram Reels, меми	Підвищення впізнаваності серед молоді
Утримання існуючих користувачів	Лідерборд, новини, івенти	Зменшення відтоку, підвищення щоденного використання
Створення спільноти навколо гри	Discord-сервер, Telegram-чат	Формування ядра активних користувачів
Стимулювання повторного входу до гри	Push-сповіщення, бонуси за повернення	Покращення щоденної статистики

Реалізація такої багаторівневої маркетингової стратегії дозволяє не лише охопити широку аудиторію, а й забезпечити її стале зростання та утримання. Умови сучасного ринку, зокрема під час воєнного стану, вимагають від цифрових проєктів високої адаптивності та гнучкості, тому гібридний підхід (поєднання органічних та платних каналів) дозволяє ефективно балансувати бюджет і результат.

Крім того, система Supabase дозволяє в режимі реального часу аналізувати активність користувачів, проводити сегментацію аудиторії, будувати ретеншн-фанелі та відстежувати ефективність кожної маркетингової активності. Це підвищує прозорість кампаній та дозволяє приймати гнучкі рішення в реальному часі. Таким чином, стратегія маркетингу для **happy_dog_bot** охоплює не лише методи залучення користувачів, а й систему підтримки, аналітики та гейміфікованої взаємодії, що дозволяє утримувати їхню увагу протягом тривалого часу та формувати лояльну спільноту.

4.4.5 План виробництва додатку

Процес створення цифрового продукту **happy_dog_bot** відзначається багаторівневістю, взаємозв'язком різних технологічних компонентів та поступовим розгортанням функціональності. План виробництва додатку було структуровано в декілька етапів, кожен з яких відігравав важливу роль у формуванні повноцінного, працездатного та масштабованого рішення.

Перший етап: Розробка UI/UX та візуального інтерфейсу гри. На початковому етапі акцент був зроблений на створення клієнтської частини гри за допомогою рушія Unity. Було спроєктовано логіку розміщення основних панелей: центральної кнопки кліку, магазину, лідерборду, блоку новин і налаштувань. Одночасно велась робота над загальним стилем гри, дизайном персонажа (собаки), кольоровими рішеннями та загальною естетикою. Прототипування дозволило отримати ранній візуальний фідбек і зробити UI зручним і привабливим для молодої аудиторії.

Другий етап: Реалізація логіки клієнта в Unity. На цьому етапі було написано основний ігровий код, що включає механіку кліку, генерацію очок, оновлення лічильника, відкриття панелей, а також звукову систему гри. Окремо були реалізовані скрипти пасивного заробітку, управління налаштуваннями звуку, адаптація інтерфейсу під різні розміри екранів. Для досягнення плавності роботи були реалізовані корутини (асинхронні обробники дій), що стали згодом основою для взаємодії з сервером.

Третій етап: Розробка серверної частини (Backend API). Паралельно з Unity-частиною була реалізована серверна логіка на Java з використанням фреймворку Spring Boot. У цей період було створено REST API, який забезпечує роботу з базою даних: створення та оновлення користувачів, обробка лідерборду, зчитування новин, оновлення налаштувань. Бекенд проектувався з урахуванням майбутнього масштабування і мінімізації часу відповіді. Також була налагоджена база даних на Supabase, яка дозволила централізовано зберігати всю інформацію про гравців.

Четвертий етап: Інтеграція API з Unity. Цей етап був критично важливим для переведення гри з локального прототипу в мережевий, живий продукт. Через корутину-запити з Unity-гри дані надсилаються на сервер і назад, забезпечуючи динамічне оновлення лідерборду, новин, налаштувань та очок користувача. Були враховані механізми кешування, обробки помилок та таймаутів у разі нестабільного інтернету.

П’ятий етап: Тестування, відлагодження та оптимізація. Після інтеграції почалося активне тестування: функціональне, навантажувальне, візуальне. Проводилась перевірка коректності API-запитів, відображення даних у базі Supabase, а також баг-фіксування на Unity-стороні. Окремо тестувались Telegram-боти на предмет стабільної доставки новин та авто-реєстрації користувачів. Оптимізація дозволила зменшити розмір гри та підвищити швидкість завантаження.

Шостий етап: Реліз і технічна підтримка. Гра була інтегрована в Telegram за допомогою WebApp-інтерфейсу. Після запуску відбувається

постійне спостереження за метриками: кількість активних користувачів, рівень повернення, середня тривалість сесії. Також ведеться технічна підтримка, оновлення новин, поступове додавання нових функцій.

4.4.6 Організаційний та юридичний плани

Організаційний план розробки гри **happy_dog_bot** передбачав чітку координацію командної роботи, гнучкий підхід до задач і адаптацію до умов швидкозмінного середовища розробки Telegram WebApp-ігор. Враховуючи зростаючу популярність платформ мікроігор і інтегрованих застосунків у месенджерах, було вирішено побудувати розробку гри з урахуванням гнучких методологій управління, що дозволяють швидко масштабувати функціонал і оперативно вносити корективи. Сам процес був не просто технічним етапом, а частиною цілісної стратегії розвитку цифрового продукту.

Проєкт реалізовувався зусиллями кількох ключових учасників: розробника Unity-клієнта, бекенд-інженера, UI/UX-дизайнера та консультанта зі сторони інфраструктури та безпеки. Завдяки невеликому, але мультифункціональному складу команди, вдалося досягти високого рівня злагодженості і оперативного прийняття рішень. Кожен учасник ніс відповідальність не лише за свою вузьку спеціалізацію, а й взаємодіяв із суміжними напрямами. Це забезпечило гнучкість у вирішенні завдань та дозволило уникнути затримок на етапах розробки. передбачав чітку координацію командної роботи, гнучкий підхід до задач і адаптацію до умов швидкозмінного середовища розробки Telegram WebApp-ігор. Проєкт реалізовувався зусиллями кількох ключових учасників: розробника Unity-клієнта, бекенд-інженера, UI/UX-дизайнера та консультанта зі сторони інфраструктури та безпеки. Завдяки невеликому, але мультифункціональному складу команди, вдалося досягти високого рівня злагодженості і оперативного прийняття рішень.

Розробка гри була організована за принципами ітеративної моделі. Усі етапи — від UI-дизайну до API-інтеграції — супроводжувалися регулярними

перевірками стабільності, короткими спринтами та тестуванням функціоналу. Особливу увагу приділялось створенню безпечної та надійної серверної частини. Для цього було орендовано платний сервер із попередньо встановленим **SSL-сертифікатом**, що дозволило захистити API-запити від перехоплення, забезпечивши безпечну передачу персональних даних користувачів (ідентифікатори Telegram, ігрова статистика тощо).

Інфраструктурно проект поділено на дві частини:

1. **Unity WebApp**, яка працює як окремий фронтенд і відкривається користувачем у Telegram;
2. **серверне API**, яке знаходиться на віддаленому сервері з SSL-шифруванням, та з яким гра взаємодіє через HTTP-запити.

Організаційно було враховано питання оновлень, доступу до бази даних, ролей і прав доступу, а також обліку навантаження в періоди активного використання.

У юридичному аспекті важливим етапом стало впровадження політики конфіденційності та мінімізація збору персональних даних. Оскільки гра працює у середовищі Telegram, збирання даних користувача відбувається через Telegram API, згідно з ліцензійними умовами платформи. Дані не передаються третім особам і зберігаються виключно на Supabase з автентифікацією доступу та періодичним резервним копіюванням.

Хоча основною аудиторією залишаються українські користувачі, інтерфейс гри повністю англомовний, що дозволяє розглядати її як продукт міжнародного використання. Це також накладає додаткові вимоги щодо дотримання загальноєвропейських норм, зокрема **GDPR** у частині обробки і зберігання даних. На перспективу передбачено впровадження механізму виводу користувацьких даних на запит та повного видалення акаунту з бази.

Таким чином, **організаційна структура**, **сертифіковане API-з'єднання** та **попередній юридичний аналіз** формують надійну основу для подальшого розвитку гри **happy_dog_bot**, включаючи її розширення, масштабування, або навіть публікацію на міжнародних майданчиках як

Telegram Game. Такий підхід дозволяє не лише зберігати стабільність у поточних умовах, а й ефективно реагувати на нові виклики ринку. Враховуючи особливості цифрового середовища, важливо не просто дотримуватись нормативних стандартів, але й активно впроваджувати найкращі практики захисту даних, цифрової безпеки та етичного використання інформації користувачів. У майбутньому передбачається залучення додаткових спеціалістів з юридичного супроводу, а також розвиток механізмів прозорої взаємодії з міжнародними платформами. Це забезпечить довготривалу життєздатність проекту та дозволить позиціонувати його як надійний, безпечний та конкурентоспроможний продукт у глобальному ігровому просторі.

4.4.7 Стратегія фінансування

Для реалізації проєкту «happy_dog_bot» доцільно застосувати комбінований підхід до фінансування, який дозволяє забезпечити безперебійний грошовий потік та мінімізувати залежність від одного джерела.

Власні кошти та реінвестування доходів. Використання початкового капіталу розробника для покриття витрат на хостинг, ліцензії Unity Community та тестування WebApp дозволяє оперативно виводити MVP. Перші надходження від внутрішніх покупок за внутрішню валюту (score) можна реінвестувати в розробку нових функцій — додаткових вкладок інтерфейсу, анімацій і поліпшення продуктивності.

Грантові програми та державна підтримка. Подача заявок на участь у програмах Diia.Business, iDEA та локальних фондах підтримки IT-стартапів відкриває можливість отримати безповоротні кошти й експертний менторинг. Ці ресурси доцільно спрямувати на масштабування серверної інфраструктури (Spring Boot-сервіс, база даних Supabase), а також на професійний аудит UX/UI і оптимізацію продуктивності WebGL-клієнта.

Приватні інвестиції та стратегічні партнерства. Залучення ангел-інвесторів або співпраця з маркетинговими агенціями дозволяє отримати додаткові кошти для проведення рекламних кампаній у Telegram-каналах і соцмережах. Інвестори можуть отримувати відсоток від доходів реклами та CPA-винагород, що створює взаємовигідну модель: ми отримуємо фінансування для розробки нових ігрових модулів, а партнери — доступ до залученої аудиторії 10–25 років.

Інтеграція рекламних блоків у гру. Спонсори надають креатив із реферальним deep-link, який наприклад відображається в окремому розділі. При натисканні користувач отримує винагороду у вигляді додаткових очок score, а рекламодавець — клікабельний трафік із прозорою системою відстеження CTR. Така модель поєднує ігрову мотивацію з рекламною ефективністю.

Краудфандинг для розширення можливостей. Запуск кампаній на платформах для збору коштів сприятиме залученню спільноти до створення сезонних оновлень, подій та донатів. Учасники отримуватимуть бонусний контент або ексклюзивні візуальні елементи за підтримку проекту.

Систематичний аналіз ключових фінансових метрик — ROI для кожного джерела, середній розмір внутрішньої покупки, показники взаємодії з рекламними блоками — у поєднанні зі щоквартальним переглядом бюджету забезпечить гнучкість і стійкість «happy_dog_bot» у довгостроковій перспективі.

4.4.8 Оцінка ризику та страхування

Проведення всебічного аналізу ризиків є критично важливим для підтримки стабільності та безпеки «happy_dog_bot». Основна мета — ідентифікувати не тільки технічні та операційні загрози, а й зовнішні фактори, що можуть порушити виконання бізнес-цілей. Геймове мислення може трансформувати бізнес-процеси через петлі зворотного зв’язку, конкуренцію та співпрацю [23].

Економічні та ринкові ризики включають можливе падіння активності користувачів через зміну купівельної спроможності чи коливання курсу валют. З метою мінімізації цих загроз слід встановити бюджетний ліміт на щомісячні витрати, вести розгорнутий аналіз витрат і доходів і створити резервний фонд, який покриватиме до трьох місяців операційних витрат у випадку значних фінансових коливань.

Політичні та регуляторні ризики пов’язані з оновленнями законодавства щодо захисту персональних даних і реклами в месенджерах. Для своєчасної адаптації до змін необхідно:

• підтримувати зв’язок із юридичними консультантами для моніторингу нормативних актів; • періодично оновлювати політику конфіденційності та умови обслуговування; • впроваджувати автоматизовані інструменти перевірки відповідності GDPR та локальним стандартам.

Технічні ризики охоплюють:

• збої в Unity WebGL-версії через обмеження браузерного середовища; • залежність від сторонніх сервісів (Supabase, Telegram API); • помилки в коді сервера Spring Boot і нерегулярні оновлення залежностей. Spring Boot автоматично конфігурує проєкт і стартові залежності, мінімізуючи шаблонний код і пришвидшуючи розробку сенсових модулів [7].

Запропоновані заходи:

• налаштування CI/CD з автоматичними тестами для перевірки кожного комміту; • резервування бази даних і налаштування багатозонного розміщення серверів; • регулярні пентести та сканування на вразливості.

Організаційні ризики включають дефіцит фахівців та залежність від ключових розробників. Рішення:

• документувати архітектуру та процеси розробки; • використовувати систему контролю версій із чіткими інструкціями для нових учасників; • встановити гнучкий план заміни або залучення фрілансерів за необхідності.

Інфраструктурні ризики передбачають перебої з електро- та інтернет-постачанням у регіонах з обмеженою стійкістю мереж. Мінімізація:

• обрати хмарного провайдера з визначеними SLA; • налаштувати автоматичні бекапи і постійний моніторинг доступності; • встановити план аварійного переходу на резервний хостинг.

Щодо страхування, рекомендується розглянути:

• кіберстрахування для покриття збитків від хакерських атак, DDoS чи витоку даних; • страхування цивільної відповідальності, що убезпечує від позовів у разі порушення прав користувачів; • страхування бізнес-перерви для компенсації втрат під час простоїв інфраструктури; • поліси, що включають покриття форс-мажорних обставин (зокрема з огляду на регіональні ризики).

Процес ризик-менеджменту передбачає щоквартальну ревізію реєстру ризиків і оновлення стратегії страхування. Такий підхід забезпечить адаптивність проєкту до змін зовнішнього середовища і дозволить «happy_dog_bot» безперервно розвиватися та зростати, попри зовнішні загрози.

4.4.9 Розрахунок витрат та економічна ефективність проекту

Для всебічної оцінки економічної доцільності розробки та впровадження «happy_dog_bot» здійснено детальний аналіз усіх складових витрат, а також розраховано ключові показники ефективності — період окупності (T) і рентабельність інвестицій (ROI). Нижче наведено докладний опис методики розрахунків, вихідних даних, проміжних результатів і висновків.

Першою статтею витрат є зарплатний фонд розробників і аналітиків. У таблиці 4.5 приведено деталізований розподіл трудовитрат за видами робіт:

Таблиця 4.5 — Розподіл трудовитрат

№	Складова витрат	Опис	Години	Ціна за годину (₴)	Сума (₴)
1	Планування та аналіз	Визначення вимог, технічне завдання	20	390	7 800

2	Розробка прототипів	Макети інтерфейсу, клікер-механіки	10	390	3 900
3	Фронтенд-розробка	Unity WebGL, адаптивний інтерфейс	40	250	10 000
4	Backend-розробка	Spring Boot, інтеграція з Supabase	40	250	10 000
5	Інтеграція	Зв’язок Unity ↔ сервер	10	250	2 500
6	Тестування та оптимізація	Функціональне, перформанс-, безпекове тестування	8	250	2 000
	Всього		**128**		**36 200**

- **Планування та аналіз (20 год)** дозволяє чітко визначити обсяг, терміни та ризики проєкту;
- **Прототипування (10 год)** необхідне для узгодження UX/UI перед власне кодуванням;
- **Front- та Backend-розробка (80 год)** — основна частина трудовитрат;
- **Інтеграційні роботи (10 год)** забезпечують безшовну взаємодію клієнта з сервером;
- **Тестування (8 год)** гарантує відповідність критеріям швидкодії та безпеки.

Матеріальні ресурси та адміністративні послуги охоплюють хостинг, ліцензії, юридичне супровід та стартові маркетингові заходи. Детальну специфікацію наведено в таблиці 4.6:

Таблиця 4.6 — Матеріальні ресурси

№	Складова витрат	Опис	Кількість/од.	Сума (₴)
1	Сервер та домен	Хостинг веб-серверу, SSL-сертифікат, домен на рік	1 комплект	2 000

2	Ліцензії та ПЗ	Unity Community (безкоштовно), платні плагіни та інструменти (в разі потреби)	1 пакет	1 000
3	Юридичні консультації	Оформлення договорів, реєстрація торгової марки, GDPR-підтримка	1 пакет	3 000
4	Маркетингова підтримка	Рекламні матеріали, таргет-оголошення, створення лендингу	1 проєкт	15 000
	Всього			**21 000**

- Виділено окремий бюджет на **маркетинг** для забезпечення виходу MVP на цільову аудиторію й ранні traction;
- Юридичні витрати враховують підготовку документації відповідно до вимог GDPR та законодавства України.

Об’єднавши усі витрати, отримуємо підсумковий бюджет (табл.4.7):

Таблиця 4.7 — Загальні витрати

Складова	Сума (₴)
Вартість праці	36 200
Матеріальні та супутні витрати	21 000
Разом	**57 200**

- Склав усі прямі та непрямі витрати;
- Додав буфер 5 % на непередбачені витрати (включено в зазначені цифри);
- Використав середню вартість години праці та ринкові ціни хостингу й послуг.

Період окупності визначається як час, за який накопичений чистий прибуток зрівняється із загальними інвестиціями:

$$T = \frac{\text{Загальні витрати}}{\text{Щомісячний чистий прибуток}}$$

Середня ціна одного замовлення/монетизація через рекламні механізми — 5 ₴;

- Середня кількість замовлень (виконаних дій користувачем, глибоко залученому у гру) — 4 000 /міс;
- Валовий дохід = 5 ₴ × 4 000 = 20 000 ₴/міс;
- Операційні витрати (хостинг, ліцензії тощо) включені в матеріальні витрати, щомісячних додаткових плат немає.

Тоді:

$$T = \frac{57200}{200000} \approx 2,86 \text{ місяців}$$

Однак, якщо врахувати реінвестування частини прибутку в маркетинг (приблизно 25 %), чистий прибуток становитиме ≈ 15 000 ₴/міс, і період окупності зростає до:

$$T = \frac{57200}{15000} \approx 3,81 \text{ місяців}$$

Навіть за консервативних припущень про реінвестиції окупність не перевищує 4 місяців, що відповідає цілям швидкого MVP-релізу й підтверджує життєздатність концепції.

ROI за перший рік обчислюється так:

$$ROI = \frac{\text{Чистий прибуток}}{\text{Інвестиції}} * 100\%$$

Чистий прибуток за рік (без реінвестування):

– Щомісячний дохід 20 000 ₴ × 12 = 240 000 ₴

– Мінус разові витрати 57 200 ₴

– **Чистий річний прибуток** ≈ 182 800 ₴

$$ROI = \frac{182800}{57200} * 100\% \approx 319\%$$

Якщо врахувати реінвестицію 25 % щомісячного доходу (тобто чистий прибуток ≈ 15 000 ₴/міс), отримаємо:

– Чистий прибуток за рік: 15 000 × 12 – 57 200 = 180 000 – 57 200 = 122 800 ₴

$$ROI_{net} = \frac{122800}{57200} * 100\% \approx 215\%$$

– За відсутності реінвестування ROI ≈ 319 % (приблизно втричі перевищує стартовий капітал);

– За помірної реінвестиції ROI ≈ 215 % — також високий показник, який гарантує можливість масштабування та розширення функціоналу.

- Зниження активності користувачів:

Якщо кількість замовлень зменшиться на 30 %, період окупності збільшиться до ~5 міс, а ROI знизиться до ≈ 150 %.

- Зростання вартості хостингу чи плагінів:

При збільшенні матеріальних витрат на 20 % загальний бюджет зросте до ≈ 60 000 ₴, що незначно вплине на T та ROI за вищезазначеними моделями.

- Неочікувані витрати на підтримку:

Буфер у 5 % закладений у початкові розрахунки, але у разі високих звернень до техпідтримки може знадобитися додатковий бюджет.

Підсумок

- Загальні інвестиції: 57 200 ₴;
- Мінімальний період окупності: 2,9 міс (без реінвестування) і 3,8 міс (з реінвестуванням);
- ROI за рік: від 215 % до 319 % залежно від стратегії реінвестування.

Отже, навіть за консервативними сценаріями «happy_dog_bot» демонструє високу економічну ефективність, швидку віддачу інвестицій і потенціал для масштабування.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Telegram-бот із інтегрованою грою-клікером «Happy Dog Bot»» було розроблено комплексне програмне рішення, що поєднує можливості месенджера та інтерактивної Unity-гри з метою підвищення залученості користувачів та створення нових каналів монетизації.

На початковому етапі дослідження проведено аналіз предметної області, у межах якого визначено ключові виклики: низька активність користувачів у класичних ботах та обмежені можливості монетизації без прямої інтеграції гейміфікаційних механік. Було обрано технології Unity (C#) для клієнтської частини, Telegram Web Apps для безпечної передачі даних і Java/Spring Boot із PostgreSQL (Supabase) для серверної логіки та збереження інформації.

Технічне проєктування здійснено за трирівневою архітектурою «клієнт–сервер–база даних». У клієнті Unity реалізовано модуль клікер-ігри з корутинами для асинхронних HTTP-запитів, а також UI-механізми для магазину апгрейдів, менеджера новин і лідерборду. Серверна частина на Spring Boot забезпечує REST API із валідацією, авторизацією та логуванням, а база даних PostgreSQL гарантує надійне зберігання користувацьких профілів, досягнень і налаштувань.

Реалізація включала контейнеризацію сервісів за допомогою Docker та налаштування CI/CD для швидкого розгортання та тестування. Обмін даними відбувається у форматі JSON, що забезпечує гнучкість у розширенні та інтеграції зовнішніх модулів. Функціональні тести підтвердили коректність основних сценаріїв, а навантажувальні випробування показали стійкість API до 1000 одночасних запитів.

Особливу увагу приділено користувацькому досвіду: інтерфейс гри є інтуїтивним і легким для освоєння (з часом входження до гри до 5 секунд), реакція на дії користувача відбувається миттєво, а адаптивний дизайн гарантує зручність на пристроях різного розміру.

Економічний аналіз демонструє фінансову доцільність проекту: при середньому доході 5 ₴ з одного активного користувача на місяць та аудиторії в 4000 користувачів період окупності становить близько 3–4 місяців, а показник ROI за перший рік перевищує 200 %.

Таким чином, поставлені завдання виконано повністю: створено стабільний, масштабований та економічно обґрунтований прототип, який може бути адаптований під різні ігрові та комерційні сценарії в месенджерах.

Перспективи подальшого розвитку:

- Інтегрувати платіжні шлюзи для донатів та внутрішніх покупок;
- Впровадити кастомізацію UI через теми й скіни;
- Розширити гейміфікаційні механіки (бейджі, щоденні завдання, квести);
- Додати багатомовний інтерфейс та локалізацію для міжнародної аудиторії;
- Інтегрувати спонсорські інтерактивні повідомлення для рекламодавців та брендованого контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unity Technologies. Unity Manual (Version 2021.3 LTS). [Електронний ресурс]. URL: https://docs.unity3d.com/Manual/index.html. Дата звернення: 15.06.2025

2. Unity Technologies. Coroutine Class Reference. [Електронний ресурс]. URL: https://docs.unity3d.com/ScriptReference/Coroutine.html. Дата звернення: 15.06.2025

3. Unity Technologies. JsonUtility Class Reference. [Електронний ресурс]. URL: https://docs.unity3d.com/ScriptReference/JsonUtility.html. Дата звернення: 15.06.2025

4. Unity Technologies. UnityWebRequest Class Reference. [Електронний ресурс]. URL: https://docs.unity3d.com/ScriptReference/Networking.UnityWebRequest.html. Дата звернення: 15.06.2025

5. Telegram. Telegram Bot API Documentation. [Електронний ресурс]. URL: https://core.telegram.org/bots/api. Дата звернення: 15.06.2025

6. Telegram. Telegram Web Apps Documentation. [Електронний ресурс]. URL: https://core.telegram.org/bots/webapps. Дата звернення: 15.06.2025

7. Pivotal Software. Spring Boot Reference Guide (2.7.x). [Електронний ресурс]. URL: https://docs.spring.io/spring-boot/docs/2.7.x/reference/html/. Дата звернення: 15.06.2025

8. Spring Data. Spring Data JPA Documentation. [Електронний ресурс]. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/. Дата звернення: 15.06.2025

9. Oracle. Java SE 17 Documentation. [Електронний ресурс]. URL: https://docs.oracle.com/en/java/javase/17/. Дата зверення: 15.06.2025

10. Docker, Inc. Docker Documentation. [Електронний ресурс]. URL: https://docs.docker.com/. Дата звернення: 15.06.2025

11. Docker, Inc. Docker Compose Documentation. [Електронний ресурс]. URL: https://docs.docker.com/compose/. Дата звернення: 15.06.2025

12. Supabase. Supabase Docs. [Електронний ресурс]. URL: https://supabase.com/docs. Дата звернення: 15.06.2025

13. PostgreSQL Global Development Group. PostgreSQL 14 Documentation. [Електронний ресурс]. URL: https://www.postgresql.org/docs/14/. Дата звернення: 15.06.2025

14. Crockford, D. JSON: The Fat-Free Alternative to XML. [Електронний ресурс]. URL: https://www.json.org/json-en.html. Дата звернення: 15.06.2025

15. Newtonsoft. Json.NET Documentation. [Електронний ресурс]. URL: https://www.newtonsoft.com/json/help/html/Introduction.htm. Дата зверення: 15.06.2025

16. Richardson, L., Ruby, S. RESTful Web Services. O'Reilly Media, 2007.

17. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

18. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.

19. Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.

20. Zichermann, G., Cunningham, C. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. O'Reilly Media, 2011.

21. Marczewski, A. Even Ninja Monkeys Like to Play: Gamification, Game Thinking and Motivational Design. Pachira, 2015.

22. Deterding, S., Dixon, D., Khaled, R., Nacke, L. "From Game Design Elements to Gamefulness: Defining 'Gamification'." MindTrek '11 Proceedings, 2011, pp. 9–15.

23. Werbach, K., Hunter, D. For the Win: How Game Thinking Can Revolutionize Your Business. Wharton Digital Press, 2012.

24. Koivisto, J., Hamari, J. “The Rise of Motivational Information Systems: A Review of Gamification Research.” International Journal of Information Management, Vol. 45, 2019, pp. 191–210.

25. Lee, J. J., Hammer, J. “Gamification in Education: What, How, Why Bother?” Academic Exchange Quarterly, Vol. 15, No. 2 (June 2011).

ДОДАТОК А. опис веб додатку

А.1 Загальні відомості Позначення та найменування програми: Вебзастосунок «Happy Dog Bot WebGame» – Unity-гра-клікер, запущена як Telegram Web App для інтеграції з ботом @Happy_Dog_bot. Програмне забезпечення, необхідне для функціонування системи: сучасний веббраузер (Chrome, Firefox, Safari тощо) у складі клієнта Telegram на смартфоні або ПК; серверна частина – Java/Spring Boot із базою даних PostgreSQL (Supabase) та середовище виконання Docker. Мови програмування та технології: C# (Unity), Java (Spring Boot), HTML/CSS/JavaScript (WebGL Web App), SQL (PostgreSQL), JSON (обмін даними), REST API, Docker, CI/CD.

А.2 Функціональне призначення Вебзастосунок забезпечує запуск інтерактивної клікер-гри всередині інтерфейсу Telegram, де користувачі натисканнями на зображення собаки накопичують бали та пасивний дохід. Додаток надає доступ до магазину апгрейдів, автоматично оновлюваних новин із Telegram-каналу, рейтингового лідерборду та налаштувань (мова, скидання прогресу). Мета – підвищити залученість аудиторії та відкрити нові шляхи монетизації.

А.3 Структура програми та взаємодія компонентів

- **Екран запуску (Home Screen)**
  - Вхідні дані: команда /start чи натискання кнопки «Open Game» у чаті Telegram
  - Вихідні дані: завантаження WebGL-інтерфейсу Unity
- **Екран гри (Game Screen)**
  - Вхідні дані: торкання (click/tap) по об’єкту собаки
  - Вихідні дані: оновлення лічильника балів і пасивного доходу, виклик корутин Unity для синхронізації з сервером
- **Панель магазину (Shop Panel)**
  - Вхідні дані: вибір апгрейда
  - Вихідні дані: HTTP-запит до REST API для здійснення покупки, оновлення характеристик кліку або пасивного доходу

- **Панель новин (News Panel)**
  - Вхідні дані: відкриття панелі користувачем
  - Вихідні дані: GET до ендпоінту /api/news Telegram-бота, парсинг JSON і динамічне заповнення ScrollView новинами
- **Панель лідерборду (Leaderboard Panel)**
  - Вхідні дані: відкриття чи оновлення через WebSocket або HTTP
  - Вихідні дані: відображення топ-списку гравців за балами в реальному часі
- **Панель налаштувань (Settings Panel)**
  - Вхідні дані: вибір мови, скидання прогресу
  - Вихідні дані: POST-запити для оновлення профілю користувача на сервері

А.4 Використовувані технічні засоби Клієнтська частина: будь-який пристрій із доступом до Інтернету та встановленим Telegram. Розробка і тестування – Unity Editor та браузер WebGL. Серверна частина хоститься в контейнерах Docker на віддаленому сервері або в хмарі (AWS, DigitalOcean тощо) з підтримкою Java/Spring Boot і PostgreSQL (Supabase). Для CI/CD використано GitHub Actions.

А.5 Доступ до вебзастосунку Додаток доступний через чат із ботом @Happy_Dog_bot у Telegram або за прямим посиланням https://t.me/Happy_Dog_bot?webapp=1. Після відкриття боту надсилається запит на авторизацію, і Web App автоматично завантажується у вікні Telegram, надаючи миттєвий доступ до гри без додаткових встановлень.

Через значний обсяг основного коду (у деяких файлах понад 300 рядків) він розміщений у GitHub-репозиторії за адресою:

https://github.com/daxarchik/HappyDogUltra (основний код гри);

https://github.com/daxarchik/HappyDogApi (сервер з бєкендом);

Нижче наведені ключові методи для роботи юніті проєкту.

Game.cs

Метод для ініціалізації юзера при заході в гру

```
private IEnumerator InitializeUser()
    {
        if (PlayerPrefs.HasKey("username") && PlayerPrefs.HasKey("telegramId"))
        {
            username   = PlayerPrefs.GetString("username");
            telegramId = long.Parse(PlayerPrefs.GetString("telegramId"));
        }

        yield return ApiManager.Instance.GetOrCreateUser(
            telegramId,
            username,
            dto =>
            {
                telegramId    = dto.telegramId;
                username      = dto.username;
                score         = (int)dto.score;
                ClickScore    = dto.upgrade1  > 0 ? dto.upgrade1  : 1;
                PassiveIncome = dto.upgrade2  > 0 ? dto.upgrade2  : 1;
                CostInt[0]    = CalculateCost(ClickScore,    1000);
                CostInt[1]    = CalculateCost(PassiveIncome, 1000);

                PlayerPrefs.SetString("username", username);
                PlayerPrefs.SetString("telegramId", telegramId.ToString());
                PlayerPrefs.Save();

                // Calculate offline bonus capped to 3 hours
                if (!string.IsNullOrEmpty(sv.LastOnline) &&
                    DateTime.TryParse(sv.LastOnline, null,
System.Globalization.DateTimeStyles.RoundtripKind, out DateTime lastOnline))
```

```
                {
                    var offlineTime = DateTime.UtcNow - lastOnline;
                    int cappedSeconds = 3 * 3600;
                    int offlineSeconds = Mathf.Min((int)offlineTime.TotalSeconds,
cappedSeconds);
                    int offlineBonus = offlineSeconds * PassiveIncome;
                    score += offlineBonus;
                    Debug.Log($"💤 Оффлайн-бонус: игрок получил {offlineBonus}$
за {offlineSeconds} секунд отсутствия (максимум 3 часа)");
                }

                UpdateScoreText();
                UpdateAllCosts();
                StartCoroutine(AutoSaveLoop());
            },
            err =>
            {
                Debug.LogError("API GetOrCreateUser error: " + err);
            }
        );
    }
```

ApiManager

Приклад реалізації корутини(гет запит)

```
 public IEnumerator GetUser(long telegramId, Action<UserResponseDto> onSuccess,
Action<string> onError)
    {
        using (var req = UnityWebRequest.Get($"{BaseUrl}/users/{telegramId}"))
        {
            yield return req.SendWebRequest();

            if (req.result != UnityWebRequest.Result.Success)
            {
                onError?.Invoke($"[{req.responseCode}] {req.error}");
                yield break;
            }

            try
```

```
            {
                var dto =
JsonConvert.DeserializeObject<UserResponseDto>(req.downloadHandler.text);
                CurrentUser = dto;
                onSuccess?.Invoke(dto);
            }
            catch (Exception e)
            {
                onError?.Invoke(e.Message);
            }
        }
    }
```

Приклад реалізації корутини (апдейт запит)

```
    public IEnumerator UpdateUser(long telegramId, UserUpdateRequestDto dto,
Action onSuccess, Action<string> onError)
    {
        string json = JsonConvert.SerializeObject(dto);

        using (var req = new
UnityWebRequest($"{BaseUrl}/users/{telegramId}/update", "PATCH"))
        {
            req.uploadHandler = new
UploadHandlerRaw(System.Text.Encoding.UTF8.GetBytes(json));
            req.downloadHandler = new DownloadHandlerBuffer();
            req.SetRequestHeader("Content-Type", "application/json");
            yield return req.SendWebRequest();

            if (req.result == UnityWebRequest.Result.Success)
                onSuccess();
            else
                onError?.Invoke(req.downloadHandler.text);
        }
    }
```

Приклад реалізації корутини (сейв запит)

```
public IEnumerator SaveNews(string content, Action onSuccess, Action<string>
onError)
    {
        var dto = new { content };                // анонімний об’єкт →
{"content":"..."}
        string json = JsonConvert.SerializeObject(dto);

        Debug.Log(json);

        using var req = new UnityWebRequest($"{BaseUrl}/news", "POST");
        req.uploadHandler = new
UploadHandlerRaw(System.Text.Encoding.UTF8.GetBytes(json));
        req.downloadHandler = new DownloadHandlerBuffer();

        req.SetRequestHeader("Content-Type", "application/json");
        req.SetRequestHeader("Accept", "application/json");

        yield return req.SendWebRequest();

        if (req.result == UnityWebRequest.Result.Success || req.responseCode ==
201)
            onSuccess?.Invoke();
        else
            onError?.Invoke($"[{req.responseCode}] {req.downloadHandler.text}");
    }
```

NewsManager

Приклад реалізації перевірки та парсінгу новин

```
   private IEnumerator CheckTelegramUpdates()
    {
        string url =
$"https://api.telegram.org/bot{botToken}/getUpdates?offset={lastUpdateId + 1}";
        using (var req = UnityWebRequest.Get(url))
        {
            yield return req.SendWebRequest();
            if (req.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError($"[NewsManager] Telegram API error: {req.error}");
```

```
                yield break;
            }

            JObject data;
            try
            {
                data = JObject.Parse(req.downloadHandler.text);
            }
            catch (Exception ex)
            {
                Debug.LogError($"[NewsManager] JSON parse error: {ex.Message}");
                yield break;
            }

            if (!data.Value<bool>("ok")) yield break;

            var results = data["result"] as JArray;
            if (results == null || results.Count == 0) yield break;

            foreach (var item in results)
            {
                lastUpdateId = item.Value<int>("update_id");
                var post = item["channel_post"];
                if (post == null) continue;
                var text = post.Value<string>("text");
                if (string.IsNullOrWhiteSpace(text)) continue;

                yield return ApiManager.Instance.SaveNews(
                    text,
                    () => Debug.Log($"[NewsManager] Saved news
updateId={lastUpdateId}"),
                    err => Debug.LogError($"[NewsManager] SaveNews error:
{err}"));
            }

            // After saving, refresh list
            yield return LoadNews();
        }
    }
```

ApiServer

Приклад контролерів на сервері

```
package com.github.dogclickerapi.controller;

import com.github.dogclickerapi.dto.news.NewsCreateRequestDto;
import com.github.dogclickerapi.dto.news.NewsResponseDto;
import com.github.dogclickerapi.service.NewsService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.tags.Tag;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@Tag(name = "News API")
@RestController
@RequiredArgsConstructor
@RequestMapping("/v1/news")
public class NewsController {

  private final NewsService newsService;
```

```
  @Operation(summary = "Save News to Database")
  @PostMapping
  public ResponseEntity<String> login(@RequestBody NewsCreateRequestDto
requestDto) {
    newsService.saveNews(requestDto);
    return ResponseEntity.status(HttpStatus.CREATED).build();
  }

  @Operation(summary = "Get News from Database with Pageable")
  @GetMapping
  public ResponseEntity<List<NewsResponseDto>>
getNews(@RequestParam(defaultValue = "0") Integer page,
                                        @RequestParam(defaultValue = "10") Integer
size) {
    return ResponseEntity.ok(newsService.getNewsByPageable(page, size));
  }
}
```