Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти − бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

Тема роботи:
«Розробка та розгортання сесійної Telegram MiniApp гри з впровадженням платежів»

Виконав студент гр. КН-21 _____ Литвиненко О. С.

Керівник _____ Харламенко В. Ю.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студентові групи КН-21 *Литвиненку Олександру Сергійовичу*

1. **Тема кваліфікаційної роботи:** *«Р*озробка та розгортання сесійної Telegram MiniApp гри з впровадженням платежів*»*

затверджено наказом по університету № *254с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *06.06.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 64с., презентація у Microsoft PowerPoint (14 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 ст. викладач Харламенко В.Ю.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.03.25*
2	*Розділ 1*	*05.04.25*
3	*Розділ 2*	*01.05.25*
4	*Висновки*	*25.05.25*
5	*Оформлення кваліфікаційної роботи*	*28.05.25*
6	*Підготовка презентації та графічного матеріалу*	*20.05.25*
7	*Підготовка доповіді до захисту*	*05.06.25*

6. Дата видачі завдання: *28.01.2025р.*

Керівник _____ /Харламенко В. Ю./

7. Запевнення: *Я, Литвиненко Олександр Сергійович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Литвиненко О. С./

АНОТАЦІЯ

Литвиненко О. С. Розробка та розгортання сесійної Telegram MiniApp гри з впровадженням платежів: кваліфікаційна робота бакалавра : 122 – Комп'ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 64 с.

Інтерактивні застосунки у месенджерах, зокрема в Telegram, стають дедалі популярнішими як формат для реалізації розважального контенту. У роботі розглянуто повний цикл розробки Telegram MiniApp-гри з підтримкою ігрових сесій, інтеграцією криптовалютних платежів і реалізацією багатокористувацького геймплею в режимі реального часу.

Метою роботи є розробка архітектури, реалізація та розгортання мікросервісної системи Telegram MiniApp гри, що включає підтримку сесійної гри, керування балансом, інтеграцію з платіжним сервісом CryptoBot і використання сучасних інструментів веброзробки.

У вступі обґрунтовується актуальність теми роботи, сформульована мета та завдання для її досягнення.

У першому розділі проаналізовано принципи побудови Telegram ботів, існуючі приклади реалізації платіжних та ігрових ботів, а також сформульовано основні технічні вимоги до цільового застосунку.

Другий розділ присвячено проєктуванню архітектури мікросервісної системи гри, моделюванню структури бази даних, побудові Telegram MiniApp інтерфейсу та інтеграції платіжної системи.

Практична цінність роботи полягає в побудові повнофункціонального фреймворку Telegram MiniApp, який може слугувати основою для створення комерційних інтерактивних застосунків з підтримкою платіжної логіки.

Ключові слова:

TELEGRAM MINIAPP, КРИПТОПЛАТЕЖІ, МІКРОСЕРВІСНА АРХІТЕКТУРА, WEBSOCKET, NESTJS, PRISMA, REDIS, DOCKER, CRYPTOBOT.

ЗМІСТ

ВСТУП

Стандартні шляхи до реалізації багатокористувацьких застосунків можуть змінюватися з часом, від складних програм під Windows до онлайн інструментів, все для зручності використання користувачами. Не так давно стали популярні чат боти в різних месенджерах, таких як: Viber, Telegram, WhatsApp та інші. Ця зручність обумовлення швидкою доступністю то застосунку та легкості у використанні, так як така реалізація імітує спілкування. Краще за все з різноманітністю функціоналу виконано в Telegram, є багато інструментів для: інструменти для оплати, зручне маніпулювання з командами, внутрішні додатки та багато іншого.

Предметом цієї роботи є повноциклічна розробка та розгортання сесійної гри в рамках Telegram MiniApp з реалізацією платіжного функціоналу. «Сесійна» у контексті означає, що геймплей складається з конкретного ігрового сеансу, кожен триває певний час, після чого результат фіксується, і гравець може переходити до наступного змагання або перегляду статистики. Даний підхід дозволяє організовувати як одноразові змагання між друзями, так і регулярні турніри зі збереженням рейтингів.

Актуальність теми обумовлена кількома факторами. По-перше, зростання популярності Telegram MiniApp створює нові можливості для розробників легких і швидких рішень, сумісних із більшістю платформ. По-друге, користувацький інтерес до ігор у месенджерах постійно зростає, що підтверджується збільшенням кількості встановлених бото-додатків та ігор у Telegram.

Таким чином, виконання поставлених завдань забезпечить створення повноцінного функційного демонстраційного проєкту Telegram MiniApp гри з оплатами, який може стати основою для подальших досліджень і комерційних продуктів у сфері інтерактивних месенджер-додатків.

РОЗДІЛ 1. АНАЛІЗ ФУНКЦІОНАЛЬНИХ І ТЕХНІЧНИХ ВИМОГ ДО БОТА СЕСІЙНОЇ MINIAPP-ГРИ

1.1 Загальні принципи роботи Telegram бота

Спочатку потрібно прояснити що таке чат бот. Чат-бот – це невелика програма в месенджері, здатна виконувати різні завдання при взаємодії з користувачем. Ця програма функціонує не як звичайний користувач, а як програмний агент, що взаємодіє через API.

Telegram Bot API – набір HTTP-запитів, який дозволяє надсилати та отримувати повідомлення, клавіатури, файли, а також обробляти callback-події.

Логіка використання визначається у вигляді діалогу: користувач, який ініціює взаємодію – telegram сервери, що отримують запити пересилають їх на сервер розробника – сервер бота, де реалізована логіка обробки повідомлень, виконання бізнес-операцій і формування відповіді.

Існує два основних способи взаємодії з Telegram Bot API:

Long Polling – це спосіб отримання оновлень від Telegram-серверів, за якого сервер бота постійно надсилає запити до Telegram Bot API з метою перевірки наявності нових повідомлень або подій. Якщо повідомлення є – API повертає їх у відповіді. Якщо повідомлень немає, запит «зависає» на певний період (до 60 секунд), після чого завершується, і бот надсилає новий запит. Цей метод є кращим для тестування.

Webhook – Telegram надсилає запити безпосередньо на визначений розробником URL. Цей метод забезпечує кращу продуктивність і швидший час відгуку, але складніший у використанні та потребує SSL сертифікат для домену. Це вже повноцінне продакшн рішення.

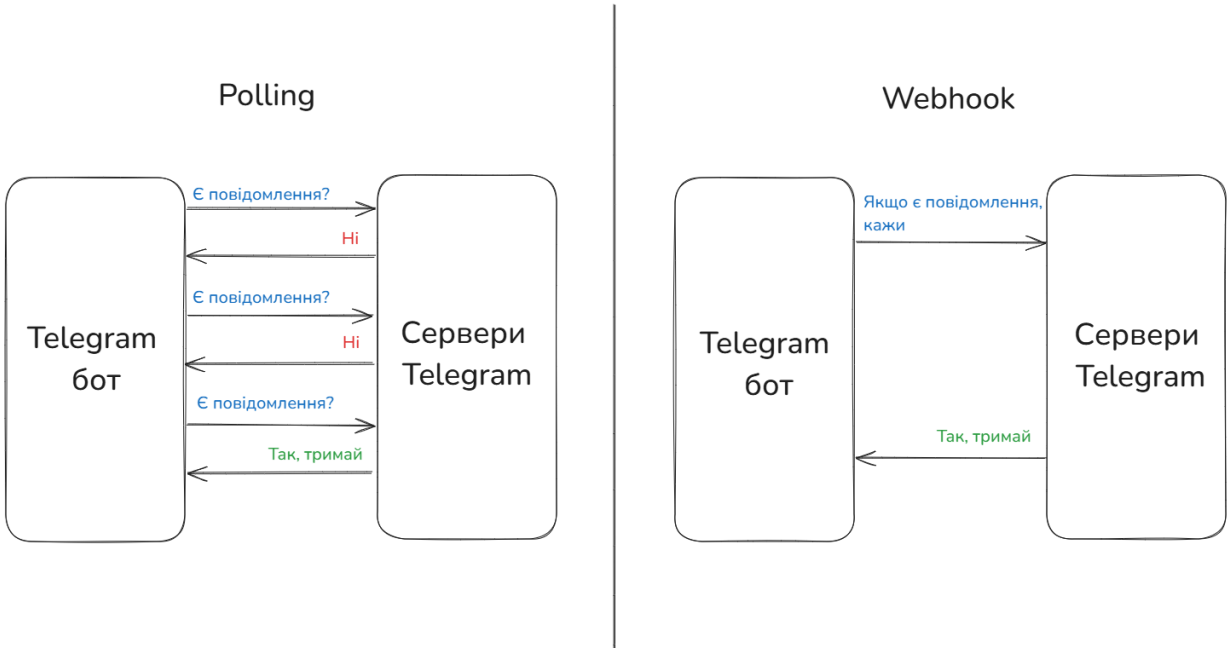

Рисунок 1.1 – Різниця між Polling та Webhook

У сучасних проєктах Telegram боти дедалі частіше виступають не як прості скрипти для автоматизації, а як повноцінні модулі взаємодії з бізнес-логікою серверної частини, базами даних, сторонніми сервісами та системами оплати. Саме тому правильне розуміння принципів їхньої роботи є ключовим для успішної реалізації MiniApp-застосунків.

Кожен бот створюється через BotFather – офіційного бота, який генерує токен авторизації, необхідний для доступу до API. Користувач надсилає команду /start, після чого обирає опцію створення нового бота. BotFather запитує ім’я бота (відображається в Telegram) та унікальний юзернейм, що має завершуватися на «bot». Після цього він надає токен доступу – спеціальний ключ, який дозволяє програмі взаємодіяти з Telegram Bot API.

На основі цього токена розробник налаштовує серверну частину, яка буде отримувати та обробляти повідомлення. У коді бота розробник програмує логіку обробки вхідних подій – це можуть бути текстові команди, натискання кнопок, callback-запити або взаємодія з WebApp. За потреби бот може звертатися до баз даних, зовнішніх API або внутрішніх сервісів. Бота можна

написати на будь якій мові програмування, яка може оброблювати зовнішні API та для який розробники месенджеру створили зручні бібліотеки.

Після реалізації та тестування логіки бот розгортається на постійному сервері або хмарній платформі. У випадку використання Webhook розробник реєструє HTTPS-адресу свого сервера через відповідний метод Telegram API. Бот переходить у стан очікування подій і може бути опублікований для кінцевих користувачів.

1.2 Аналіз сучасних Telegram ботів

Для того щоб написати дійсно конкурентоздатний застосунок, потрібно зробити аналіз ринку Telegram ботів. Визначемо основні плюси та недоліки таких рішень. Проаналізуємо боти на властивості цікавості та використання системи оплати.

Hamster Kombat – дуже популярна Telegram-гра у жанрі clicker. Гравці торують роль «гаманця-гамблера»: натискаючи на віртуального хом’яка, вони заробляють внутрішню валюту і покращують власну крипто-біржу у грі З моменту релізу у березні 2024 року вона здобула неймовірну аудиторію – понад 50 млн щоденних активних користувачів і загалом 300 млн гравців у всьому світі

- Основні функції**:** моделі play-to-earn: заробіток монет за кліки та щоденні активності; щоденні квести (комбінації карток, розшифровка азбуки Морзе тощо) для додаткових нагород; кастомізація (одяг та аксесуари для хом'яка); реферальна система для залучення друзів.
- Особливості реалізації**:** Hamster Kombat запущено як Bot із WebApp – інтерфейсом. Гра являє собою простий веб-клікер з барвистою графікою хом’яка та біржі. Інтерфейс адаптований під мобільні пристрої і відкривається через кнопку в чаті (Inline-кнопка), тому гра запускається безпосередньо в Telegram.

– Переваги**:** надзвичайна популярність і невимушений геймплей («натискай і вигравай»), що дозволяє залучати мільйони користувачів Постійні оновлення (випуск нових карт, турнірів, щоденних задач) підтримують інтерес. Інтеграція з криптовалютою: у планах – власний токен HMSTR, що мотивує гравців накопичувати монети в грі.
– Недоліки**:** геймплей досить монотонний (повторюване клацання), може швидко набридати без різноманітних активностей.

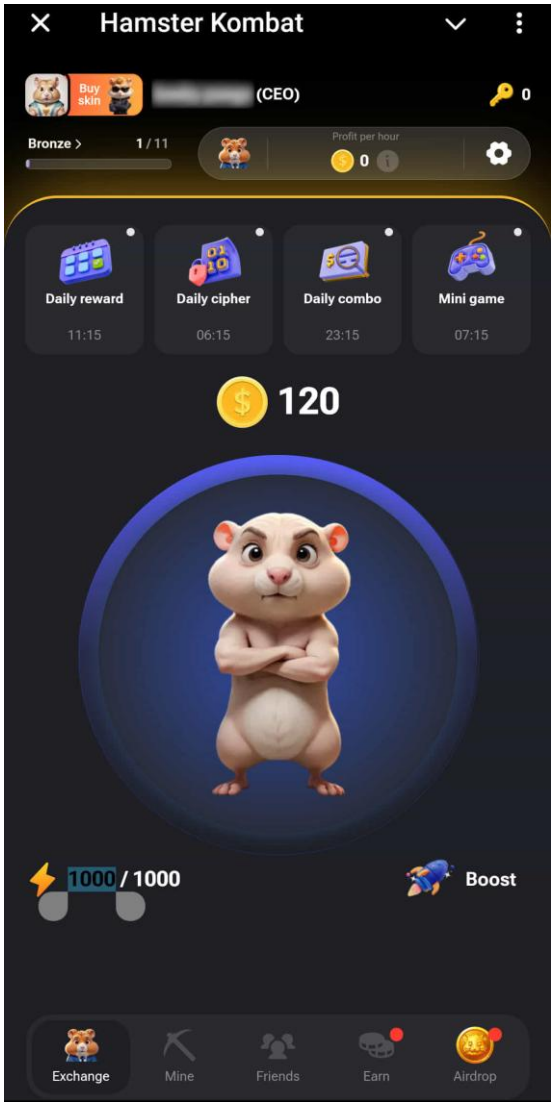

Рисунок 1.2 – інтерфейс застосунку Hamster Kombat

WayForPay Bot – зручний прийом платежів за товари чи послуги від клієнтів прямо в мессенджері Telegram.

– Основні функції: Основна функція – виставлення рахунків (інвойсів), формування електронних рахунків з інформацією про товар/послугу, суму та реквізити продавця, також може генерувати QR-коди. Прийом платежів у чаті: клієнт оплачує прямо в Telegram, бот миттєво повідомляє про успіх чи невдачу транзакції.
– Особливості реалізації: WayForPay Bot використовує вбудовані можливості Telegram для виставлення інвойсів і обробки callback-запитів от Telegram Payment API.
– Переваги: Основна перевага – що вже автоматизовано і написано, сповіщення про успішну оплату та звіти надходять безпосередньо в чаті, що знижує людський фактор і скорочує час обробки замовлень. А також присутня відповідність національному законодавству, підтримка гривні, робота з місцевими банками та НБУ.
– Недоліки: стандартна комісія WayForPay може бути вищою, ніж у деяких конкурентів. А також такий способ оплати є зручним саме для української аудиторії, але краще мати альтернативу для інших країн. А також немає механізму для отримування грошей користувачем, тільки депозит.

Рисунок 1.3 – Вигляд бота WayForPay

Crypto Bot – популярний мультивалютний криптогаманець та біржа в Telegram.

– Основні функції: дозволяє безпечно й анонімно купувати, продавати та зберігати Toncoin (TON), Bitcoin (BTC), Ethereum (ETH), USDT, BUSD та інші криптовалюти, а також поповнювати баланс криптою за допомогою віртуальних чеків і приймати оплату через електронні рахунки (чекі).
– Особливості реалізації: бот інтегрується з декількома блокчейн-мережами через API decenter.org, підтримує мультигаманці та non-custodial зберігання, використовує захищені канали Telegram для обробки запитів і підтверджень транзакцій.
– Переваги: дуже зручний механізм «Застосунок», за допомогою якого можно автоматизувати чеки та квитанції, користувачі зможуть оплачувати та виплачувати гроші. Вся оплата в криптовалюті, тому будь-хто може користуватися.
– Недоліки: користувачу, який ніколи не працював з кріптовалютами, може бути трішки складно робити оплату та отримувати кошти. Також є деяка комісія на транзакції.

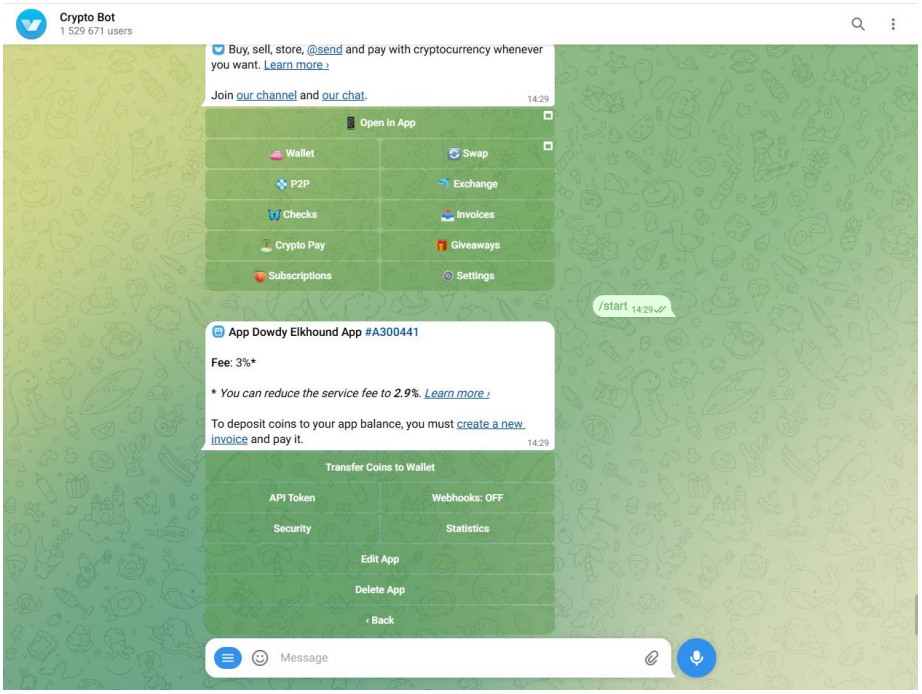

Рисунок 1.4 – Механізм «Застосунок» в CryptoBot

У висновку можна сказати що боти демонструють елементи, які будуть корисні для проєкту.

Розважальна частина: Hamster Kombat демонструє силу простого, але затягуючого клікерного геймплею. Тому сесійна реалізація може бути максимально привабливою, так як кожна сесія буде унікальною і буде затягувати своїм геймплеєм.

Фінансова частина: Crypto Bot демонструє, як у Telegram можна ефективно реалізувати мультивалютну екосистему з мінімальними зусиллями з боку розробника. Замість традиційної моделі балансу, де користувач поповнює рахунок через фіатні шлюзи, в цьому випадку працює логіка «вхід за квитанцією – вихід через чек». Гравець отримує криптовалютну квитанцію для поповнення – після її оплати монети автоматично заліковуються на баланс у грі. При бажанні вивести виграні кошти, йому генерується віртуальний чек, що дозволяє уникати складних інтеграцій із банками на початковому етапі. А депозит потім можна бути реалізовати і як через CryptoBot, так і через WayForPay.

1.3 Постановка задачі дослідження

Метою цього дослідження є створення повнофункціональної системи багатокористувацької гри, реалізованої як Telegram MiniApp, з підтримкою ігрових сесій, взаємодії з користувачами, а також інтеграцією платіжного функціоналу. Основна ідея полягає у поєднанні розважального компоненту (гри за принципом короткотривалих сесій) з повноцінною системою керування обліковими записами, балансами, технічною архітектурою та сучасними принципами побудови мікросервісних додатків.

Головним завданням стало проектування архітектури, яка дозволить легко масштабувати систему, розвивати її функціональність, оновлювати окремі компоненти незалежно одне від одного та забезпечувати стабільну взаємодію між усіма частинами програми.

Для досягнення цієї мети необхідно вирішити такі завдання:

- спроєктувати архітектуру Telegram MiniApp-гри на основі мікросервісного підходу;
- реалізувати Telegram-бота як основний засіб взаємодії з користувачем;
- побудувати API Gateway для централізованого управління запитами;
- організувати комунікацію між сервісами за допомогою брокера повідомлень RabbitMQ;
- розробити набір мікросервісів: UserService, BalanceService, LobbyService, GameService, GameWebSocket;
- створити веб-інтерфейс гри за допомогою Next.js і TailwindCSS як Telegram MiniApp;
- забезпечити обробку ігрових сесій у реальному часі через WebSocket;
- впровадити тимчасове кешування та зберігання сесій у Redis;
- інтегрувати платіжну систему через CryptoBot API з використанням Webhook;
- зберігати постійну інформацію у реляційній базі PostgreSQL з використанням Prisma ORM;
- протестувати роботу системи на основних сценаріях: реєстрація, створення сесій, обробка транзакцій, ігровий процес, завершення гри.

Цільова аудиторія проєкту – це користувачі Telegram, які зацікавлені в простих, але конкурентоздатних ігрових сесіях із можливістю вигравати. Потенційними замовниками або користувачами подібної платформи можуть бути стартапи, студії мобільних ігор або незалежні розробники, які шукають ефективну основу для створення гейміфікованих мінізастосунків у середовищі Telegram.

Проєкт робить акцент на:

- Модульності структури – кожен компонент незалежний і може розвиватися окремо;
- Високій продуктивності – за рахунок продуманої структури

- Легкості у використанні – завдяки звичному для користувачів Telegram інтерфейсу;
- Платоспроможності – зручне поповнення та виведення коштів
- Безпеці – обмеження прямого доступу до сервісів, централізована маршрутизація та контроль.

Реалізація цього проєкту продемонструє актуальні підходи до створення інтерактивних цифрових сервісів на основі сучасних вебтехнологій, мікросервісної архітектури та інфраструктурної інтеграції з Telegram, що робить дослідження важливим внеском у галузі комп'ютерних наук.

Висновки до розділу:

- Сучасні Telegram боти демонструють високий потенціал як платформа для реалізації інтерактивних застосунків, зокрема у форматі сесійних ігор, завдяки простоті використання, вбудованій підтримці платежів та MiniApp-фреймворку.
- Аналіз існуючих рішень (Hamster Kombat, WayForPay Bot, CryptoBot) дозволив виділити ключові елементи, що мають бути реалізовані у проєкті: простий, захопливий геймплей, платіжна інтеграція, масштабованість та підтримка збереження сесій.
- Постановка задачі дослідження визначила необхідність побудови багатокомпонентної мікросервісної архітектури з Telegram-ботом як точкою входу, Redis як швидким кешем, RabbitMQ для асинхронної комунікації, WebSocket для реального часу, а також підтримкою криптовалютних платежів через зовнішній API.

РОЗДІЛ 2. ПРОЕКТУВАННЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ ТЕЛЕГРАМ БОТА

2.1 Вибір технологій для реалізації проєкту

Вибір технологій для реалізації програмного забезпечення є критичним етапом, який визначає не лише ефективність розробки, але й стабільність, масштабованість і подальшу підтримку системи. В умовах побудови Telegram MiniApp-гри з підтримкою платіжного функціоналу та ігрових сесій у режимі реального часу особливо важливим є використання технологій, що підтримують асинхронну обробку, високу продуктивність та гнучку мікросервісну архітектуру.

Ключовим фронтенд-компонентом системи виступає Telegram-бот, реалізований на Node.js з використанням мови TypeScript. Він слугує точкою контакту з користувачем: обробляє повідомлення, реагує на команди, ініціює логіку створення сесій та передає необхідні дані до відповідних мікросервісів. Бот також запускає ігрову сесію у вигляді посилання на MiniApp, де і відбувається основна взаємодія.

Вже визначено зі стеком технологій, який буде використовуватися:

1. Node.js + TypeScript: Це основа всіх мікросервісів системи. Node.js – це середовище виконання JavaScript на сервері, яке забезпечує неблокуючий, подієво-орієнтований підхід до обробки запитів. Його легкість і швидкість роблять його ідеальним для систем із великою кількістю одночасних з’єднань, як у випадку з Telegram MiniApp-грою. TypeScript – надмножина JavaScript із підтримкою статичної типізації. Завдяки цьому забезпечується краща перевірка коду на етапі компіляції, полегшується масштабування проєкту, підтримка та колективна розробка.
2. Telegraf.js: фреймворк на Node.js який дозволяє гнучко керувати сценаріями спілкування через Scenes (сцени). Кожна сцена відповідає

окремому етапу взаємодії з користувачем: наприклад, введення суми поповнення, підтвердження входу в гру, очікування готовності інших гравців.

3. NestJS: Фреймворк для створення серверних додатків на Node.js з підтримкою TypeScript. Він базується на архітектурі модулів, контролерів і сервісів (аналогічно до Angular) і є ідеальним для побудови масштабованих, структурованих та тестованих додатків. У даному проєкті NestJS використовується для створення API Gateway – єдиного входу в систему, що забезпечує маршрутизацію, логіку авторизації, обробку помилок і агрегацію відповідей від інших сервісів.
4. Redis: Інструмент для швидкого кешування та тимчасового зберігання даних у пам’яті. Redis – це in-memory key-value база даних, яка дозволяє обробляти сотні тисяч операцій за секунду. У системі Redis використовується: для зберігання станів активних лобі (ID гри, гравці, готовність); як кеш профілів користувачів і результатів; для автоматичного видалення застарілих даних завдяки TTL (time-to-live).
5. PostgreSQL: Реляційна СКБД із відкритим кодом, яка забезпечує підтримку транзакцій, ACID-гарантії, складні зв’язки між таблицями, індексацію та потужний SQL. У проєкті PostgreSQL використовується як основне сховище даних для:
   - облікових записів користувачів;
   - балансів і транзакцій;
   - ігрової статистики;
   - журналу подій і логів.
6. Prisma ORM: Сучасний ORM (Object-Relational Mapping) інструмент для TypeScript/Node.js, який надає зручний типобезпечний API для взаємодії з базами даних. Prisma автоматизує генерацію SQL-запитів, забезпечує автозаповнення в IDE, спрощує роботу з міграціями, типами даних і зв’язками між сутностями.

7. RabbitMQ: Це брокер повідомлень, який дозволяє реалізовувати асинхронну подієву архітектуру. Сервіси не звертаються один до одного безпосередньо, а надсилають повідомлення в чергу, звідки вони потрапляють до споживачів. RabbitMQ дає змогу:
   - обробляти події без затримок між сервісами;
   - відокремити логіку комунікації;
   - уникнути помилок у випадку тимчасової недоступності сервісу;
   - масштабувати окремі компоненти системи незалежно.
8. Socket.IO: Бібліотека для реалізації двостороннього обміну даними в реальному часі між сервером і клієнтом через WebSocket або альтернативні протоколи. У проєкті використовується в GameWebSocket для забезпечення:
   - оновлень ходу гри в режимі реального часу;
   - взаємодії між усіма учасниками гри;
   - миттєвої синхронізації стану гри на клієнтах.
9. React.js: React – це JavaScript-бібліотека з відкритим кодом, призначена для створення користувацьких інтерфейсів, переважно односторінкових додатків (SPA). Її головна особливість – компонентний підхід, який дозволяє розділити інтерфейс на ізольовані, багаторазові елементи з власною логікою і станом. У проєкті React.js використовується для побудови фронтенду Telegram MiniApp-гри. Завдяки цьому забезпечується:
   - висока продуктивність через віртуальний DOM;
   - зручне керування станом;
   - спрощення повторного використання коду й тестування окремих частин інтерфейсу;
   - швидка інтеграція з API, WebSocket та іншими клієнтськими механізмами.

10. TailwindCSS: TailwindCSS – утилітарний CSS-фреймворк, який забезпечує швидку розробку адаптивних і сучасних інтерфейсів. Разом вони дозволяють створити легкий, швидкий і стильний ігровий інтерфейс у браузері користувача без потреби встановлення сторонніх додатків.
11. Docker: Для запуску, тестування та розгортання всієї мікросервісної інфраструктури обрано середовище Docker. Це інструмент контейнеризації, який дозволяє упакувати додатки разом з усіма залежностями в ізольовані контейнери. Такий підхід має низку важливих переваг:
    - Кожен мікросервіс може бути запущений у власному контейнері, незалежно від середовища операційної системи;
    - Забезпечується повторюваність середовища – розробник, тестувальник і продакшн-сервер використовують однакові умови виконання;
    - Спрощується масштабування, перенесення, оновлення та розгортання сервісів;
    - Полегшується розробка завдяки використанню docker-compose – інструменту для одночасного запуску кількох сервісів за єдиним конфігураційним файлом.

Таким чином, обраний набір технологій повністю відповідає вимогам до побудови сучасного, масштабованого та ефективного застосунку у форматі Telegram MiniApp. Він дозволяє реалізувати архітектуру з чітким розділенням відповідальностей, асинхронною взаємодією між компонентами, підтримкою реального часу, стабільним зберіганням даних та гнучкою системою платежів. Використання контейнеризації через Docker забезпечить зручність у розгортанні, тестуванні та супроводі системи на всіх етапах її життєвого циклу. Усі технології є сучасними, відкритими та активно підтримуваними спільнотою, що додатково підвищує надійність та довгострокову придатність обраного рішення.

2.2 Проєктування мікросервісної архітектури

У сучасній розробці програмного забезпечення одним із найпоширеніших архітектурних підходів є використання мікросервісної архітектури. Вона передбачає побудову системи як сукупності незалежних, ізольованих сервісів, кожен з яких відповідає за чітко визначену функцію або бізнес-логіку. Це протиставляється традиційному монолітному підходу, де вся логіка програми зосереджена в одному великому кодовому базисі та розгортається як єдине ціле.

Монолітна архітектура характеризується простотою початкової реалізації, але має суттєві недоліки при масштабуванні, оновленні та супроводі. З часом зростає складність керування таким застосунком: зміна одного компонента може потребувати повторної збірки та розгортання всієї системи, що створює ризик порушення інших частин.

На відміну від цього, мікросервіси дозволяють:

- розділити логіку на модулі, кожен з яких можна розробляти, тестувати й оновлювати незалежно від інших;
- масштабувати лише необхідні частини системи, не витрачаючи ресурси на весь застосунок;
- використовувати різні технології, мови програмування чи бази даних для різних сервісів (хоч у цьому проєкті використовується уніфікований стек на Node.js);
- підвищити стійкість системи, оскільки збій одного сервісу не зупиняє роботу всієї системи;
- спростити розгортання, використовуючи контейнери та оркестратори.

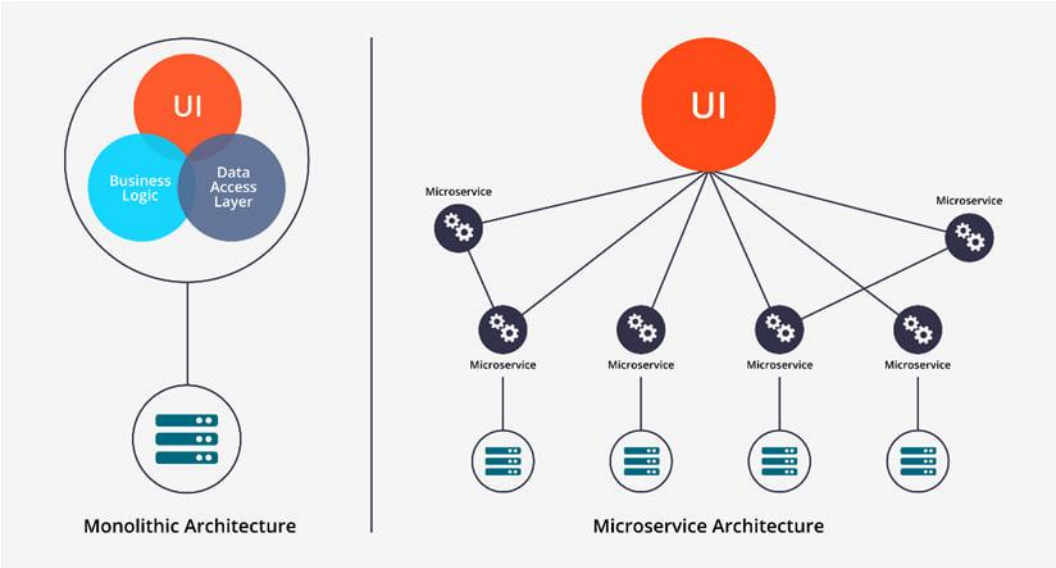

Рисунок 2.1 – Монолітна та мікросервісна архітектура

У контексті розробки сесійної Telegram MiniApp гри мікросервісний підхід є найбільш доцільним, оскільки дозволяє окремо обробляти користувачів, баланс, створення ігрових лобі, запуск сесій, WebSocket-з'єднання, платежі тощо. Така модульність значно полегшує розробку, спрощує тестування і в майбутньому дозволяє гнучко масштабувати систему – наприклад, у разі великого потоку гравців або транзакцій.

Telegram Bot (Node.js Typescript)

Бот, який в проєкті займає роль фронтенду, так як напряму працює з користувами, отримує повідомлення з користувачів і вже далі надсилає їх до сервісу, який обробляє функціонал.

API Gateway (Node.js + NestJS)

Цей сервіс є єдиною точкою входу для всіх запитів та здійснює маршрутизацію запитів до відповідних сервісів, надсилаючи відповідь. Зроблено його окремо, так як у майбутньому може бути реалізовано адміністративну панель керування, яка вже не буде телеграм ботом.

UserService (Node.js Typescript + Prisma ORM)

Сервіс, який працює з даними користувачів: ім'я, ID, дата створення акаунту, кількість зіграних ігор. За допомогою ORM Prisma працює з базою даних та виконує базові CRUD операції з даними.

BalanceService (Node.js Typescript + Prisma ORM + Webhook)

Сервіс, який працює з балансом користувача, надсилає повідомлення про оплату та отримання грошей. Також працює з Webhook, щоб отримувати повідомлення від платіжної системи про результат операції.

GameService (Node.js Typescript)

Сервіс, який відповідає за створення самої гри, прив’язує гравців, які будуть приймати участь у цій грі, а також братиме участь у передаванні даних щодо гри у Web додатку Telegram MiniApp.

CryptoBot

Зовнішній платіжний сервіс, який дозволяє користувачам поповнювати та виводити кошти в криптовалюті. Працює через API та Webhook, надсилаючи сповіщення про успішні чи невдалі транзакції. Сервіс BalanceService обробляє ці події, оновлює баланс користувача з урахуванням комісії та зберігає результат у базі даних.

GameWebSocket (Node.js + Socket.IO)

Відповідає за створення окремих сесій для ігор, після створення гри повертає посилання на MiniApp з запитаною грою по ID, а також результати гри.

GameFront (React.js + TailwindCSS)

Telegram MiniApp сайт, який буде відкриватися за ID, є основним ігровим процесом, в якій і відбувається уся гра. Гра є цифровою версією популярної настольної гри «Монополія».

Уявимо усі сервіси на діаграмі (Рисунок 2.2)

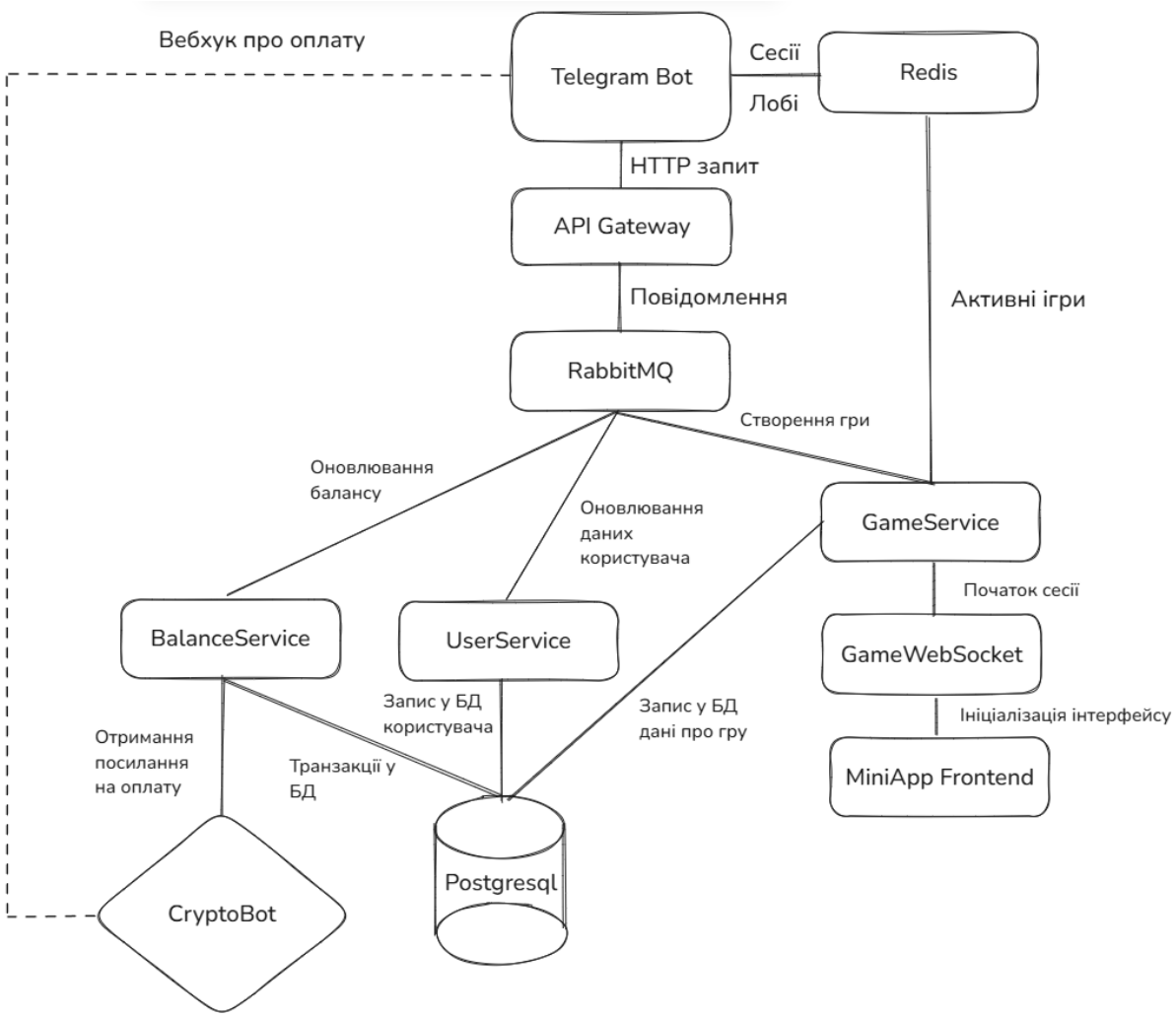

Рисунок 2.2 – Діаграма мікросервісної архітектури

Центральною точкою взаємодії виступає Telegram-бот, який через API Gateway та брокер повідомлень RabbitMQ зв’язується з окремими сервісами: керування користувачами (UserService), балансами (BalanceService), ігровим процесом (GameService) тощо. Комунікація між компонентами реалізована асинхронно, що забезпечує масштабованість і стійкість. Дані зберігаються в PostgreSQL, а Redis використовується як кеш для ігрових сесій. Платіжна інтеграція здійснюється через зовнішній сервіс CryptoBot за допомогою Webhook. GameService відповідає за створення і запуск ігрових сесій, які обробляються в реальному часі через GameWebSocket. Ігровий інтерфейс реалізовано у вигляді Telegram MiniApp, що ініціалізується при старті сесії. Усі сервіси є ізольованими, що дозволяє оновлювати або масштабувати їх

незалежно один від одного. Така архітектура забезпечує гнучкість розробки, спрощує розгортання та підтримку системи.

2.3 Проєктування структури бази даних

Архітектура бази даних побудована з урахуванням мікросервісної логіки, тобто кожен сервіс має обмежений доступ до лише тих таблиць і функцій, які йому необхідні. Завдяки цьому вдалося реалізувати чітке розділення відповідальностей і зменшити ризик взаємного впливу компонентів системи на дані.

Ключовим елементом є таблиця користувачів, у якій зберігається унікальний Telegram ID, а також характеристики, пов’язані з прогресом у грі – зокрема рівень, досвід, кількість зіграних сесій і дата створення облікового запису. Telegram ID є не лише логічним, а й технічним ідентифікатором користувача, на який зав’язано багато інших таблиць. Її опис зображенно на таблиці 2.1.

Таблиця 2.1 – Опис таблиці users

Стовпець	**Тип**	**Опис**
id	SERIAL, PK	Унікальний внутрішній ідентифікатор користувача. Використовується для системних зв’язків.
telegramId	INTEGER, UNIQUE	Telegram ID користувача, що є основним зовнішнім ідентифікатором. Унікальний у межах системи.

Продовження таблиці 2.1

Стовпець	Тип	Опис
level	INTEGER, DEFAULT 0	Поточний рівень користувача в грі. Використовується для гейміфікації.
experience	INTEGER, DEFAULT 0	Кількість досвіду, набраного користувачем. Залежно від нього може підвищуватися рівень.
gamesPlayed	INTEGER, DEFAULT 0	Лічильник усіх ігор, у яких брав участь користувач.
createdAt	TIMESTAMP, DEFAULT CURRENT_TIMESTAMP	Час створення облікового запису.

Для зберігання інформації про баланс кожного користувача створено окрему таблицю, пов’язану з користувачами за допомогою зовнішнього ключа. Варто зазначити, що після створення нового користувача за допомогою тригера автоматично ініціалізується запис у таблиці балансів, що дозволяє гарантувати послідовність і уникати ситуацій із відсутнім записом балансу. Опис зображено на таблиці 2.2.

Таблиця 2.2 – Опис таблиці balances

Стовпець	Тип	Опис
id	SERIAL, PK	Унікальний внутрішній ідентифікатор запису балансу.
telegramId	INTEGER, FK → users	Посилання на Telegram ID користувача з таблиці users. Один користувач – один баланс.

Продовження таблиці 2.2

Стовпець	Тип	Опис
value	DOUBLE PRECISION, DEFAULT 0	Поточна сума коштів, що є на балансі користувача.
createdAt	TIMESTAMP, DEFAULT CURRENT_TIMESTAMP	Час створення запису. Зазвичай створюється автоматично після реєстрації користувача через тригер.

Ще одним важливим компонентом є таблиця транзакцій, яка відстежує всі дії, пов’язані зі зміною балансу користувача. Кожен запис містить тип транзакції (поповнення або виведення коштів), суму, дату створення та посилання на відповідний запис балансу. Для забезпечення контролю консистентності даних при видаленні або зміні балансу застосовано обмеження зовнішнього ключа з каскадною поведінкою. Зображено на таблиці 2.3.

Таблиця 2.3 – Опис таблиці balance_transactions

Стовпець	Тип	Опис
id	SERIAL, PK	Унікальний ідентифікатор транзакції.
balance_id	INTEGER, FK → balances(id)	Посилання на запис балансу, до якого відноситься транзакція.
amount	INTEGER, DEFAULT 1	Сума транзакції (додатна або від’ємна, залежно від типу).
type	ENUM('DEPOSIT', 'WITHDRAW')	Тип транзакції: поповнення або зняття коштів.

Продовження таблиці 2.3

Стовпець	Тип	Опис
created_at	TIMESTAMP, DEFAULT CURRENT_TIMESTAMP	Час виконання транзакції.

Окрім фінансових операцій, у системі також передбачено структуру для зберігання активних і завершених ігрових сесій. Відповідна таблиця містить унікальний ідентифікатор гри у форматі UUID, тип гри (наприклад, дуельна або командна), список учасників, дату початку та статус активності. Важливо, що ідентифікатори користувачів зберігаються у вигляді масиву, що дозволяє ефективно працювати з динамічним складом гравців. Опис таблиці – таблиця 2.4.

Таблиця 2.4 – Опис таблиці games

Стовпець	Тип	Опис
id	UUID, PK, DEFAULT uuid_generate_v4()	Унікальний ідентифікатор гри. Генерується автоматично.
type	ENUM('DUEL_GAME', 'QUADRO_GAME')	Тип гри: дуельна (2 гравці) або командна (4 гравці).
telegramids	INTEGER[]	Масив Telegram ID учасників гри. Відповідає складу лобі.
timeofstart	TIMESTAMPTZ, DEFAULT now()	Час початку гри.
active	BOOLEAN, DEFAULT TRUE	Статус гри: активна (true) або завершена (false).

Для реалізації ролей доступу, відповідно до сервісної архітектури, у системі створено окремі облікові записи бази даних для кожного мікросервісу.

Кожному з них надано лише той мінімальний обсяг прав, який необхідний для виконання його функцій. Наприклад, сервіс обробки балансу має повний доступ лише до таблиць фінансових даних, а користувацький сервіс – до персональних даних гравців. Такий підхід мінімізує потенційні ризики в разі компрометації одного з компонентів.

Створимо діаграму для того, щоб візуально уявити архітектуру бази даних (рисунок 2.3).

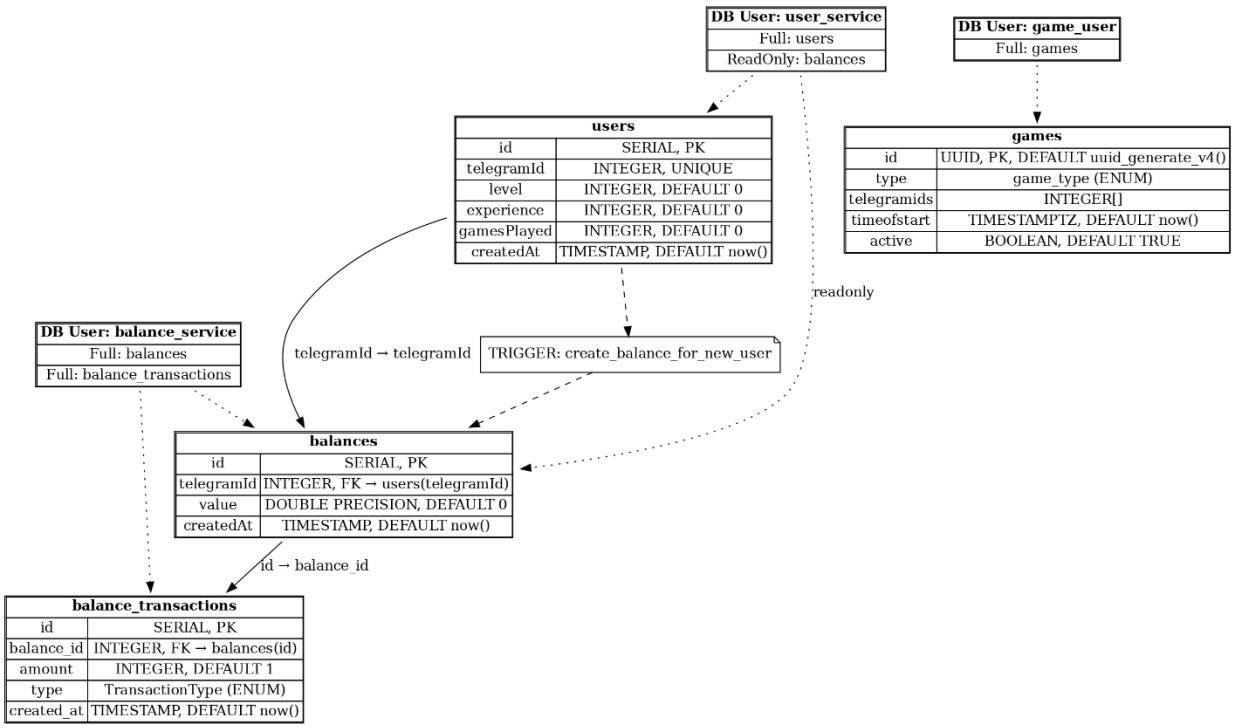

Рисунок 2.3 – Діаграма бази даних

Додаткові об’єкти:

- ENUM TransactionType – перелік допустимих типів транзакцій: 'DEPOSIT', 'WITHDRAW'.
- ENUM game_type – перелік типів гри: 'DUEL_GAME', 'QUADRO_GAME'.
- TRIGGER create_balance_for_new_user – автоматично створює запис балансу при додаванні нового користувача в таблицю users.

У результаті реалізована модель даних не лише повністю відповідає функціональним потребам системи, а й забезпечує високий рівень

узгодженості, ізоляції та безпеки. Вона легко розширюється під час розвитку системи – як за рахунок додавання нових сутностей, так і через зміну логіки взаємодії між вже існуючими.

2.4 Використання системи

Система розроблена для організації інтерактивної сесійної гри у форматі Telegram MiniApp, яка дозволяє гравцям взаємодіяти один з одним у режимі реального часу, брати участь у цифрових ігрових сесіях, керувати особистим балансом, а також здійснювати фінансові операції (наприклад, поповнення чи виведення коштів) через інтегровану криптоплатіжну систему. Усе це відбувається безпосередньо всередині Telegram, без потреби в окремому додатку.

Для забезпечення прозорого опису функціональних можливостей цієї системи була побудована діаграма сценарію користування. Вона відображає взаємодію зовнішніх користувачів (акторів) з основними сценаріями, які підтримує система. Зокрема, діаграма дозволяє чітко розділити дії, доступні звичайному гравцеві, від можливостей розробника.

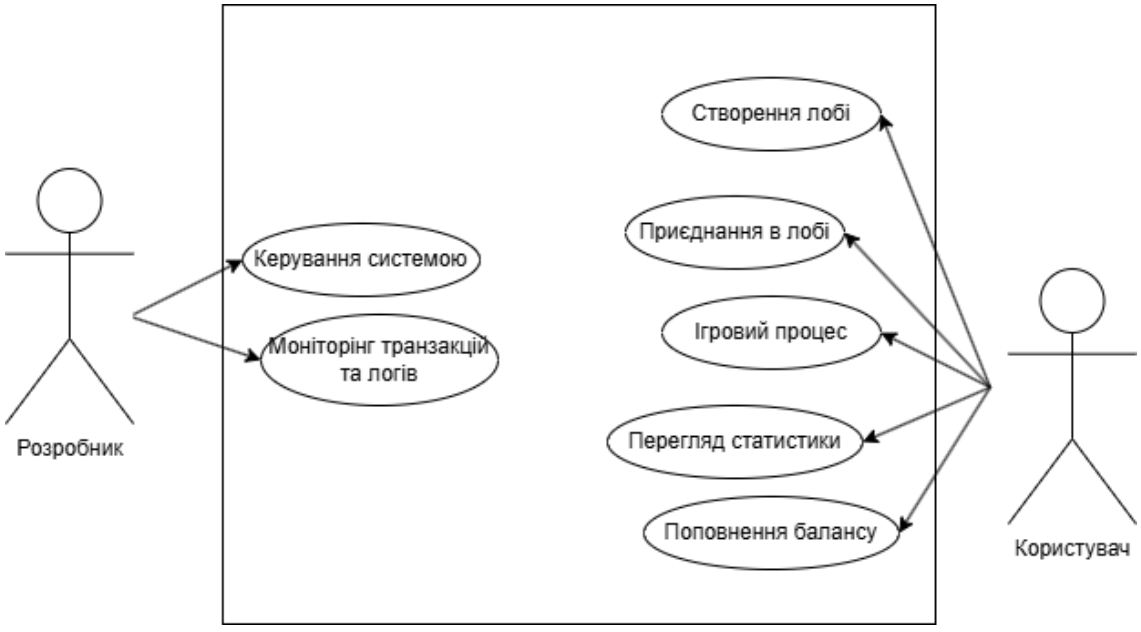

Рисунок 2.4 – Сценарій користування застосунком

Актори:

- Користувач (гравець) – основний учасник системи, що взаємодіє через Telegram MiniApp. Його мета – взяти участь у грі, управляти балансом і переглядати результати.
- Розробник – технічна особа, яка керує системою, має доступ до аналітики, транзакцій та службових журналів.

Основні сценарії користувача:

1. Створення лобі:

Гравець ініціює нову ігрову сесію, система створює у Redis новий запис, доступний для приєднання інших.

2. Приєднання в лобі:

Інші користувачі отримують посилання на сесію та входять до активного лобі, очікуючи старту.

3. Ігровий процес:

Основний цикл гри. Кожен користувач у лобі здійснює дії згідно з правилами. Дані синхронізуються через WebSocket.

4. Перегляд статистики:

Користувач має змогу переглянути свою історію ігор, перемоги, поразки, досвід, рівень тощо.

5. Поповнення балансу:

Гравець виконує фінансову операцію через інтегрований CryptoBot. Після підтвердження система оновлює баланс користувача.

Сценарії адміністратора:

1. Керування системою

Адміністратор може через окрему адмін-панель впливати на загальні параметри гри, обмеження, запуск/зупинку ігрових сервісів, створення ігрових подій.

2. Моніторинг транзакцій та логів

Доступ до перегляду платіжних подій, журналів помилок і активностей. Це необхідно для контролю стабільності системи та розв'язання конфліктних ситуацій.

2.4.1 Побудова діаграми SADT

Для візуального представлення логіки функціонування системи побудуємо SADT-діаграму (Рисунок 2.5). Такий тип діаграми (Structured Analysis and Design Technique) використовується для структурного аналізу інформаційних систем, де відображаються ключові елементи процесу: входи, управління, механізми (виконавці) та виходи.

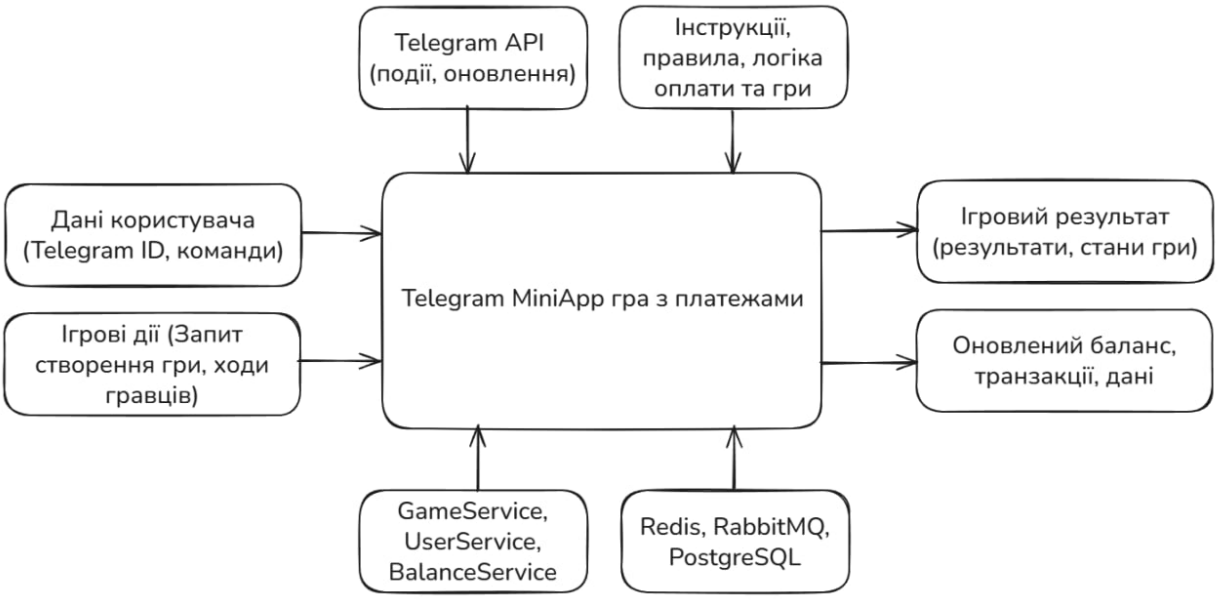

Рисунок 2.5 – SADT діаграма

Центральний блок

Telegram MiniApp гра з платежами – це головний функціональний процес, який координує всі інші підсистеми. Він включає в себе обробку команд користувача, запуск гри, управління балансом, взаємодію з базами даних, а також обробку платіжних подій. Вся логіка гри, як фронтенд, так і бекенд, об’єднана в цьому процесі.

Входи (ліворуч)

– Дані користувача (Telegram ID, команди) – це інформація, яка надходить безпосередньо від користувача через Telegram. Вона включає унікальний ідентифікатор, текстові повідомлення, натискання кнопок тощо.
– Ігрові дії (Запит створення гри, ходи гравців) – події, які ініціюються гравцем під час взаємодії з MiniApp: запит приєднатися до гри, створити сесію, зробити хід тощо.

Управління (зверху):

– Telegram API (події, оновлення) – система отримує всі події від користувачів через Telegram API. Це слугує тригером для обробки дій, запуску логіки та відповіді у вигляді повідомлень, WebApp чи WebSocket-з'єднань.
– Інструкції, правила, логіка оплати та гри – це внутрішньо закладені правила функціонування системи, що визначають умови гри, структуру сесій, обробку транзакцій, порядок оновлення балансу тощо. Також сюди належать бізнес-правила та політики системи.

Механізми (знизу):

– GameService, UserService, BalanceService – це мікросервіси, які реалізують окремі частини логіки: управління ігровими сесіями, профілями користувачів та балансами відповідно.
– Redis, RabbitMQ, PostgreSQL – це технічна інфраструктура, яка забезпечує зберігання, обмін повідомленнями та обробку даних. Redis використовується як кеш, RabbitMQ – для подій між сервісами, PostgreSQL – як основна база.

Виходи (праворуч)

– Ігровий результат (результати, стани гри) – це фінальна інформація для користувача, яка показує хід гри, переможців, очки тощо. Вона відображається у Telegram MiniApp.
– Оновлений баланс, транзакції, дані – результат обробки платіжної системи або дій гравців, які змінюють фінансовий стан користувача,

записуються до бази даних і можуть бути використані в подальших сесіях.

2.4.2 Побудова діаграми DFD

DFD-діаграма (Data Flow Diagram) – це один із ключових інструментів структурного моделювання, який відображає, як дані циркулюють у межах інформаційної системи. На відміну від use-case діаграм, які фокусуються на взаємодії користувачів із системою, DFD зосереджується на потоках даних, процесах обробки, зовнішніх сутностях і сховищах інформації (Рисунок 2.6). Ця діаграма дозволяє:

- формалізувати логіку обробки даних на всіх етапах взаємодії з користувачем;
- візуалізувати взаємозв’язки між процесами, API та базами даних;
- виявити залежності між асинхронними подіями, особливо у випадку обробки платежів;
- створити підґрунтя для побудови системної архітектури та модулів;
- використати її як технічну документацію для розробників або зовнішніх аудиторів.

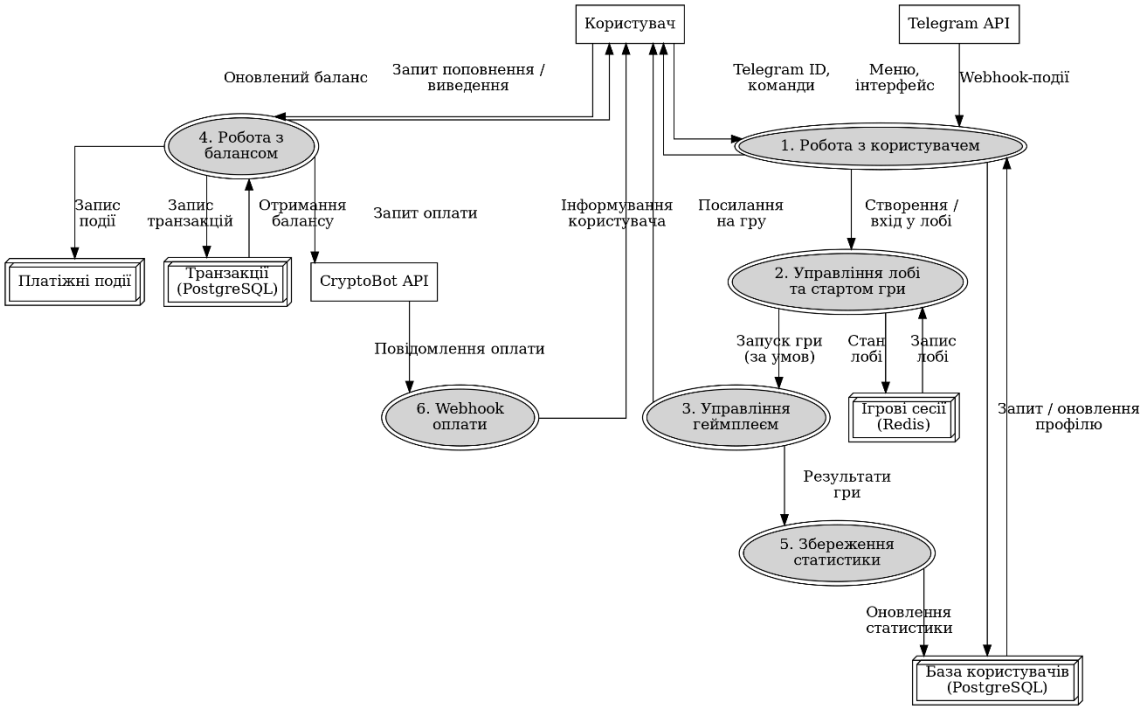

Рисунок 2.6 – Діаграма DFD

Зовнішні сутності:

– Користувач – надсилає команди через Telegram Bot, взаємодіє з ігровим інтерфейсом та переглядає результати.
– Telegram API – передає повідомлення, команди та callback-и до Telegram-бота (у нашій системі – точка входу для взаємодії).
– CryptoBot API – зовнішній платіжний сервіс, який надсилає Webhook-повідомлення про успішність або неуспішність транзакцій.

Процеси:

– Робота з користувачем – обробляє Telegram-команди, надає відповіді, звертається до бази користувачів.
– Управління лобі та стартом гри – формує або оновлює стан ігрового лобі, перевіряє готовність учасників.
– Управління геймплеєм – запускає гру, передає користувачу посилання на MiniApp, завершує сесію.
– Робота з балансом – відповідає за оновлення балансу, створення транзакцій та збереження платіжної інформації.

- Збереження статистики – обробляє ігрові результати та оновлює статистику у профілі гравця.
- Webhook оплати – приймає повідомлення від CryptoBot API та інформує користувача про результат операції.

Сховища даних:

- База користувачів (PostgreSQL) – містить профілі гравців, їхній досвід, статистику тощо.
- Ігрові сесії (Redis) – тимчасове сховище активних лобі та стану гри в реальному часі.
- Транзакції (PostgreSQL) – журнал усіх фінансових операцій.
- Платіжні події – лог Webhook-сповіщень або технічна база подій платежів.

Сценарій руху даних:

1. Користувач надсилає команду, вона потрапляє в процес Робота з користувачем.
2. Система може запитати або оновити профіль у базі користувачів.
3. Якщо гравець створює гру або приєднується до лобі – активується процес Управління лобі, який зберігає інформацію у Redis.
4. Коли учасники готові – відбувається запуск гри, і кожен отримує посилання на MiniApp.
5. Після завершення гри, результат передається в процес Збереження статистики, який оновлює дані профілю.
6. При транзакціях, дані потрапляють до процесу балансування, який фіксує дані у таблиці транзакцій та записує подію.
7. CryptoBot API надсилає результат оплати, потрапляє у Webhook-процес, який інформує користувача.

2.5 Запуск застосунку

У процесі розробки важливо забезпечити повноцінну роботу всієї мікросервісної системи навіть у локальному середовищі. Зокрема, це особливо актуально для сервісів, які взаємодіють із зовнішніми API, як-от Telegram або CryptoBot, які очікують виклики від публічно доступних адрес.

Під час розробки для ізоляції сервісів, керування залежностями та пришвидшення запуску використовуються контейнери Docker. Для кожного мікросервісу створено власний образ, а спільно всі сервіси можна запустити через docker-compose, що дозволяє миттєво отримати повноцінне середовище з піднятими Redis, PostgreSQL, RabbitMQ та іншим. Але паралельно з цим підтримується і ручний запуск – наприклад, для зручності налагодження певного компонента можна стартувати його окремо за допомогою npm run dev, а всі змінні середовища беруться з .env-файлів (Рисунок 2.7).

Щоб локальні сервіси могли працювати з Telegram API або CryptoBot API, які вимагають зовнішніх HTTPS-адрес, застосовуються тунелі.

1. Ngrok використовується для створення зовнішнього посилання на вебхук Telegram бота (наприклад, https://abcd-12-34-56-78.ngrok.io/telegram/webhook). Це дозволяє Telegram API надсилати повідомлення в локальну систему.
2. Serveo або аналогічний SSH-тунель дозволяє пробросити порти на інші сервіси:
   - URL для API Gateway (https://apimono.serveo.net)
   - URL WebSocket-сервера (https://wsmono.serveo.net)
   - Frontend MiniApp (https://appmono.serveo.net), який Telegram відкриватиме у своєму браузері.

Коли система проходить локальне тестування, настає етап розгортання. Залежно від характеру сервісу та бюджету, обираються відповідні платформи. Для backend-сервісів оптимальним варіантом є VPS-сервери (наприклад, Hetzner або DigitalOcean), де встановлюється Docker, налаштовується

обертаючий proxy (Nginx) і запуск здійснюється через docker-compose. Тут також налаштовуються домени, сертифікати HTTPS (наприклад, через Let's Encrypt) і базові механізми моніторингу.

Фронтенд (React + Tailwind) розгортається окремо – найзручніше для цього використовувати Vercel, який має інтеграцію з GitHub та автоматично оновлює сайт після кожного пушу. До того ж, Telegram MiniApp вимагає стабільної та швидкої адреси з HTTPS, тож розміщення фронтенду на спеціалізованій CDN-платформі дуже доречне.

Деякі мікросервіси можна розгорнути окремо на хмарних сервісах типу Fly.io або Render. Вони дозволяють масштабувати сервіси незалежно, автоматично забезпечують HTTPS та простий процес деплою з GitHub. Це особливо зручно для сервісів, які обслуговують вузькі завдання, наприклад, обробку платежів або WebSocket.

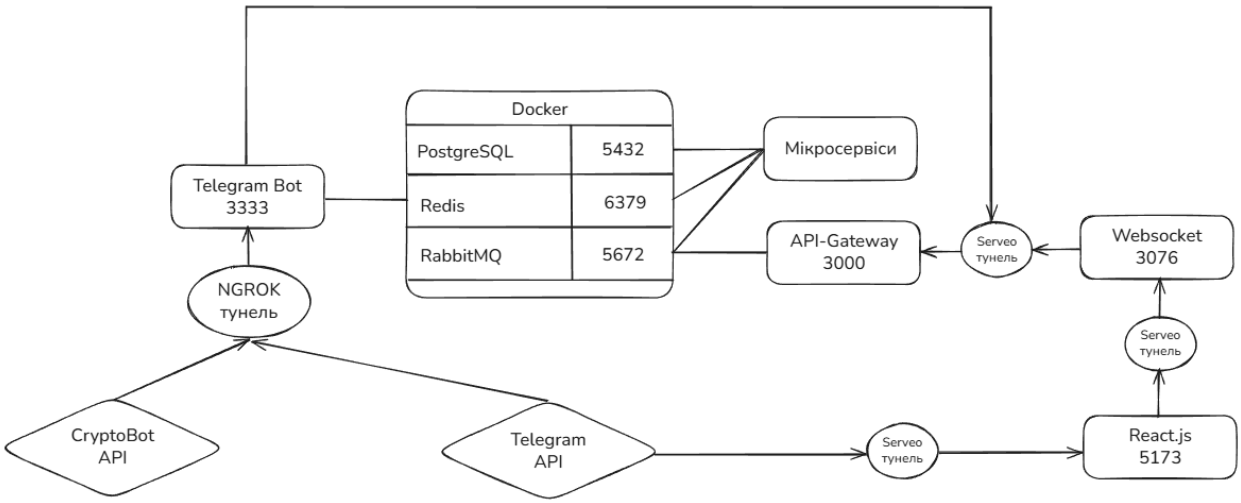

Рисунок 2.7 – Діаграма локального підключення усього проєкту

На діаграмі показано локальну інфраструктуру проєкту під час розробки та тестування. Усі основні компоненти – PostgreSQL, Redis, RabbitMQ, API Gateway і мікросервіси – запускаються в контейнерах Docker. Telegram Bot працює локально через порт 3333, отримуючи запити з Telegram API та CryptoBot API через тунель NGROK. Для доступу до фронтенду (React.js), WebSocket-сервера та API використовуються Serveo-тунелі, які дозволяють

зовнішнім системам звертатися до локальних сервісів. Такий підхід забезпечує повноцінне тестування системи до розгортання в продакшн-середовище.

2.6 Взаємодія користувача з Telegram-ботом

Telegram-бот є основним інтерфейсом взаємодії між гравцем та системою. Саме через нього користувач ініціює ігрові дії, отримує відповіді, сповіщення, переходи до MiniApp-гри та інструкції щодо користування функціоналом платформи.

Користувач починає взаємодію зі стартової команди /start, після чого бот автоматично реєструє його в системі (якщо такий профіль ще не існує), створює початковий запис у базі даних та повертає коротке привітальне повідомлення з доступним меню дій. Меню динамічне, і формується залежно від стану користувача.

Бот підтримує наступні типи взаємодії:

- текстові команди (/start) – виконують базові дії (реєстрація, запуск гри, поповнення балансу, довідка);
- інлайн-кнопки (створення/приєднання до лобі, підтвердження участі, запуск гри);
- повідомлення про події (початок гри, завершення, отримання нагороди);
- автоматичні переходи до MiniApp-фронтенду через Telegram WebApp API;
- сповіщення про статус поповнення коштів.

Система сцен дозволяє реалізувати діалог у кілька кроків, зберігаючи контекст взаємодії. Завдяки цьому бот «пам’ятає», на якому етапі знаходиться користувач, і реагує не просто на окрему команду, а на логічну послідовність дій. Наприклад, після натискання кнопки "Поповнити баланс", бот переходить у сцену введення суми, очікує відповідь, надсилає посилання на оплату – і тільки після підтвердження повертається до основного меню.

Для реалізації персоналізованої взаємодії використовується сесійне зберігання (session middleware), завдяки чому бот асоціює певні змінні (ID лобі, статус, введені дані) з конкретним користувачем у межах одного діалогу. На рівні UX це відчувається як «живий діалог», коли бот розуміє, що ви вже щось зробили до цього, і не змушує вводити інформацію повторно.

2.6.1 Команда /start

Спробуємо зайти в бота за його ім’ям @the_mono_game_bot та написати команду /start (Рисунок 2.8).

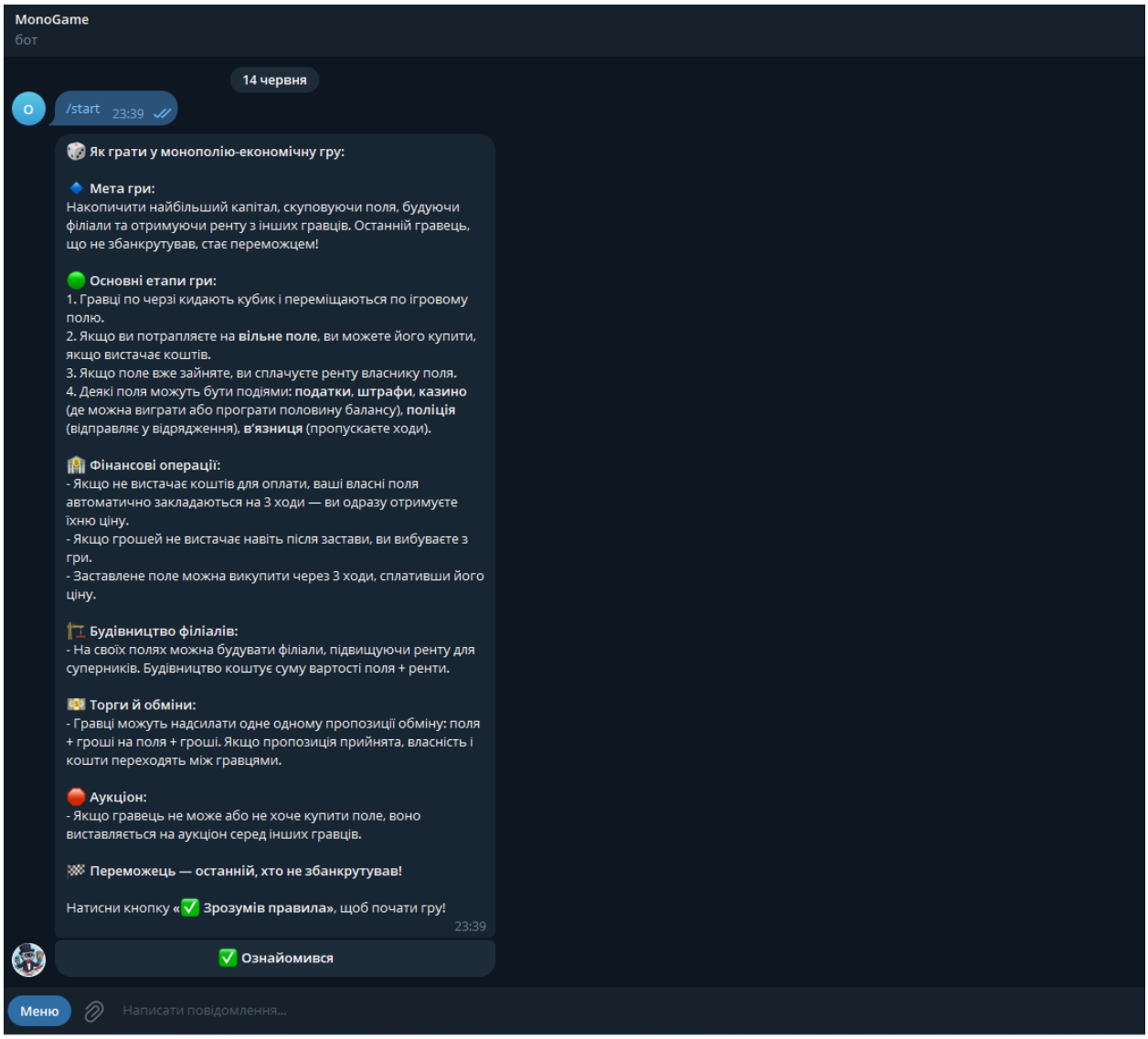

Рисунок 2.8 – Перший запуск Telegram-бота

На початку бачимо інформацію для нових гравців – правила самої гри, під повідомленням бота бачимо надпис «Ознайомився», якщо натиснути на нього – користувач автоматично переходить в сцену головного меню, натиснемо та побачимо результат (Рисунок 2.9).

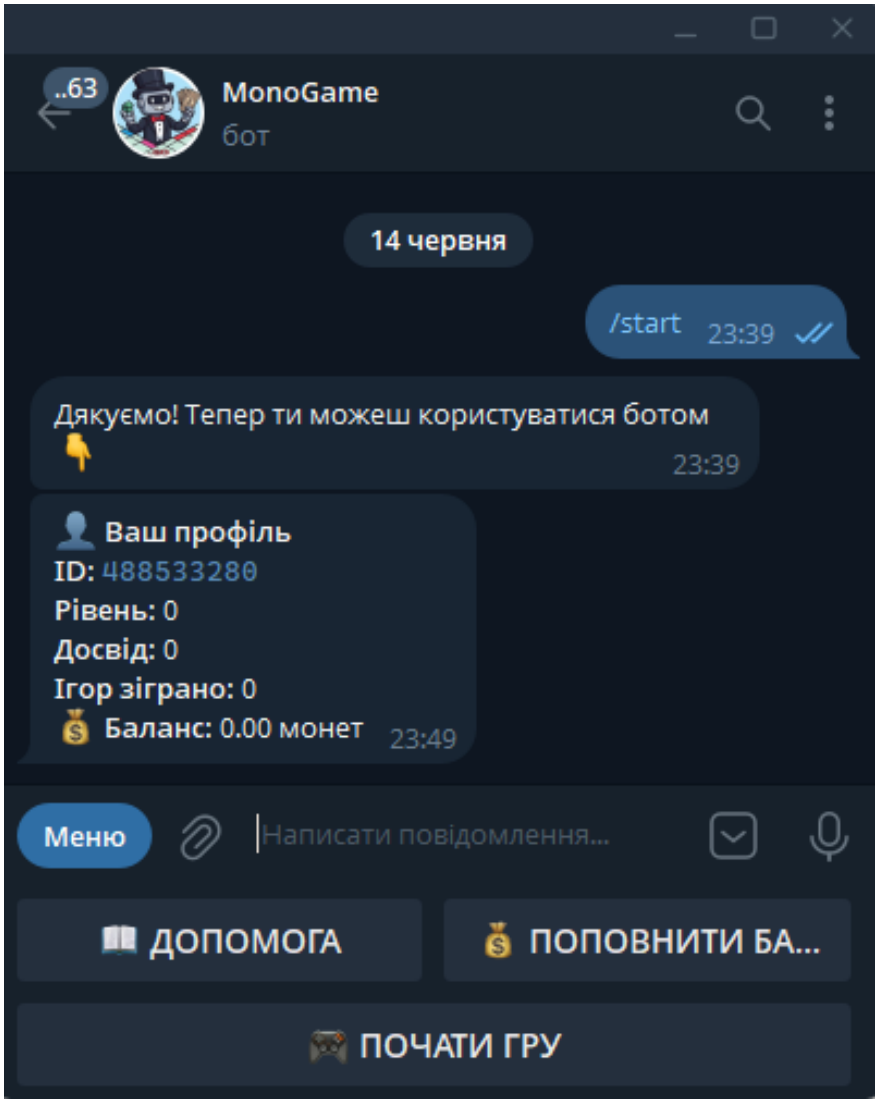

Рисунок 2.9 – Головне меню

Після успішної авторизації бот відправляє привітальне повідомлення «Дякуємо! Тепер ти можеш користуватися ботом», а також знизу блок профілю, в якому:

- ID: унікальний Telegram ID користувача;
- Рівень: ігровий рівень (початково 0);
- Досвід: кількість очок досвіду;

– Ігор зіграно: загальна статистика участі у сесіях;
– Баланс: віртуальна валюта користувача (в цьому випадку 0.00 монет).

Також бачимо інлайн-меню (Telegram WebApp-клавіатура):

– ДОПОМОГА – відкриває розділ із поясненнями щодо гри;
– ПОПОВНИТИ БАЛАНС – ініціює сцену, в якій користувач вводить суму, після чого генерується посилання на оплату через CryptoBot;
– ПОЧАТИ ГРУ – дозволяє створити нову ігрову сесію або приєднатися до вже існуючої, залежно від наявних лобі.

2.6.2 Команда ДОПОМОГА

Замість використання команд по типу «/help», бот працює саме над обробкою самих повідомлень, так як це семантично зрозуміліше .

Якщо натиснути кнопку «ДОПОМОГА» з меню, то просто отримуємо теж саме повідомлення про правила (Рисунок 2.10).

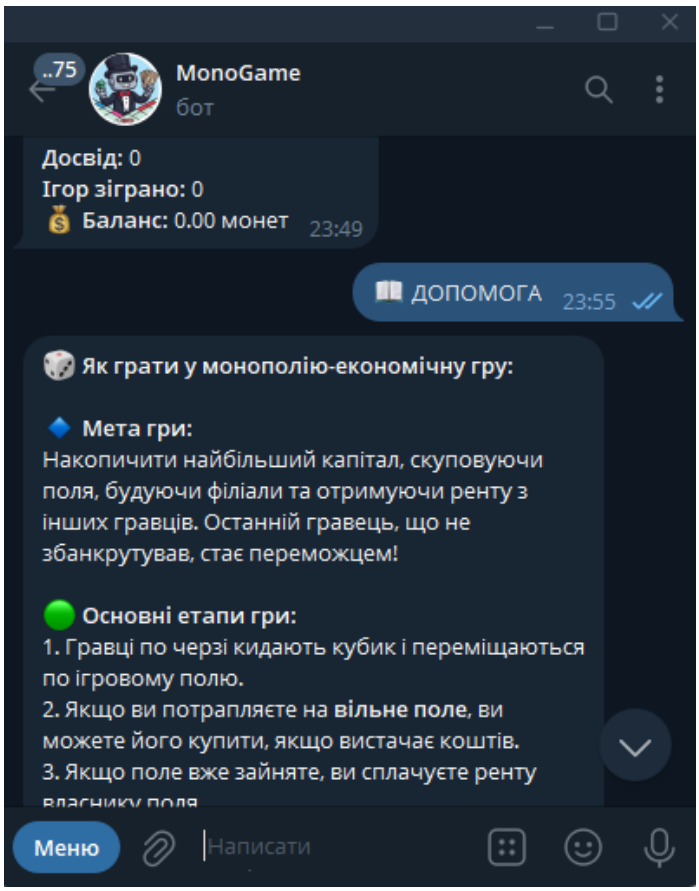

Рисунок 2.10 – Використання команди «ДОПОМОГА»

Доступ до цієї команди є суттєво постійним, завжди, коли людині потрібно згадати якусь інформацію або правила гри.

2.6.3 Команда ПОПОВНИТИ БАЛАНС

Ця команда потрібна для того, щоб використовувати платні функції застосунку, основна з них – створення нового лобі. Так як створити лобі для гри потрібно, тоді поповнимо наш баланс через USDT (Tether USDT – це токен Ethereum, який прив’язаний до вартості долара США).

Обираємо цю команду та дивимось на відповідь бота (Рисунок 2.11).

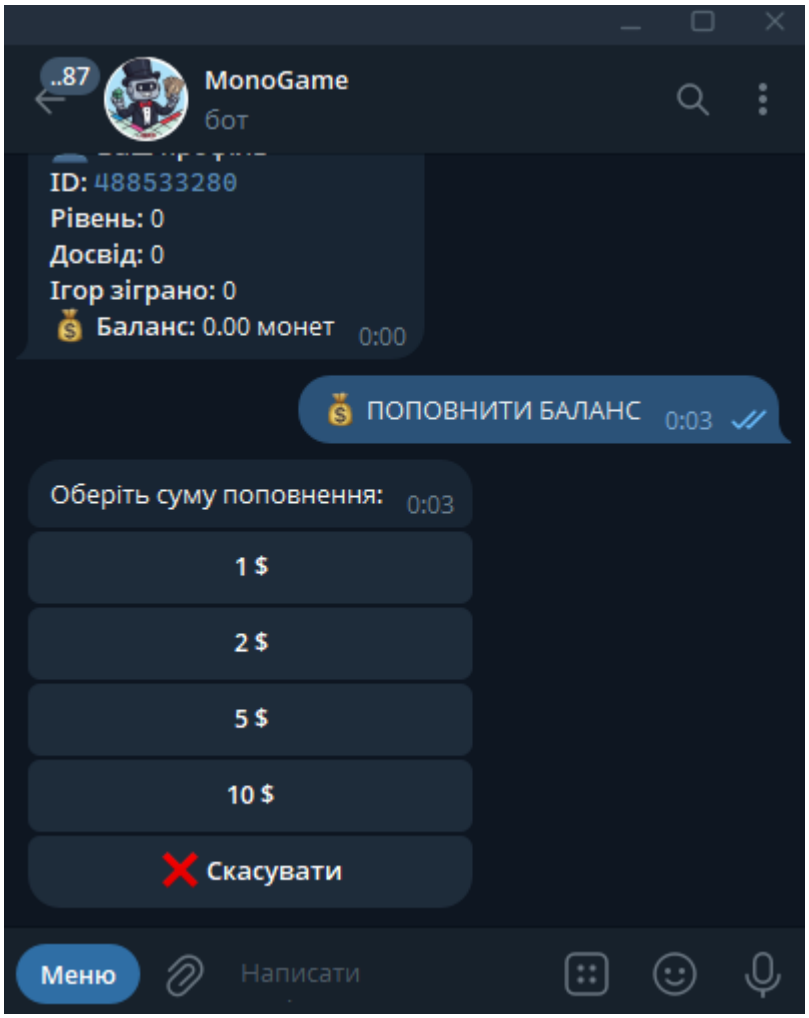

Рисунок 2.11 – Команда «ПОПОВНИТИ БАЛАНС»

Бот надає вибір, на скільки саме коштів користувач може собі поповнити баланс, оберемо найдешевший 1 USDT. Після цього користувачу повинен

прийти рахунок CryptoBot, який він повинен оплатити за посиланням (Рисунок 2.12).

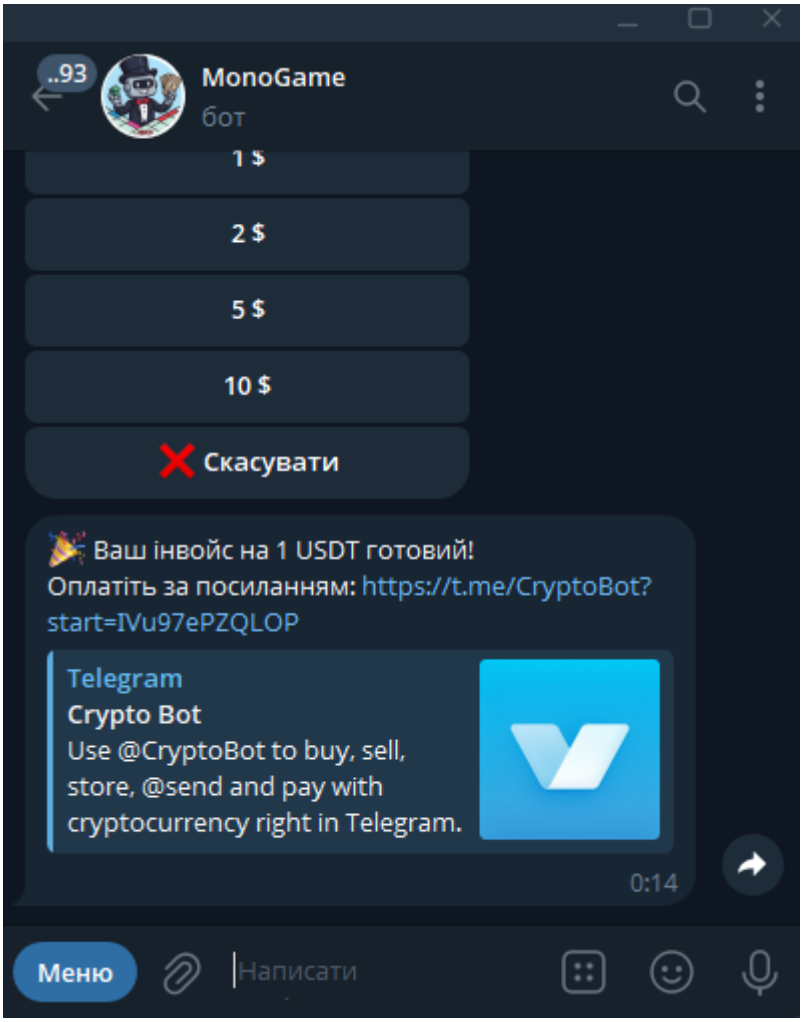

Рисунок 2.12 – Отриманий рахунок

Що буде відбуватися далі:

1. Користувач натискає на посилання: Telegram відкриває CryptoBot з уже сформованим рахунком на оплату в USDT (або іншій криптовалюті, яку підтримує бот).
2. В CryptoBot користувач підтверджує оплату: він може використати внутрішній баланс або оплатити через зовнішній гаманець (Рисунок 2.13).
3. CryptoBot надсилає Webhook у систему: Після оплати бот MonoGame отримує підтвердження через Webhook.

4. MonoGame оновлює баланс гравця: Після обробки Webhook бот повідомляє користувача про успішне поповнення та оновлює інформацію в профілі та базі даних.

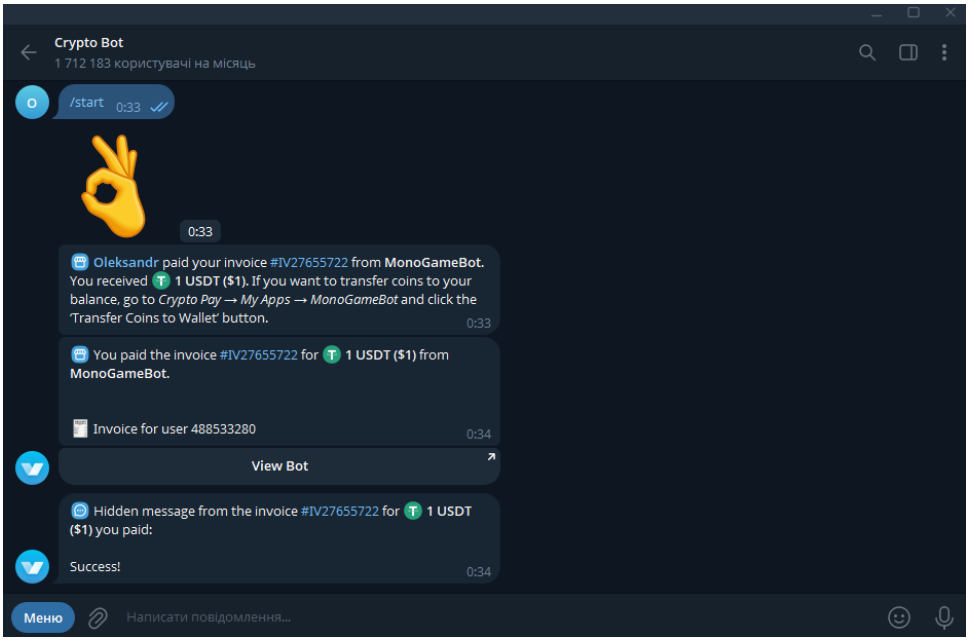

Рисунок 2.13 – Оплата рахунку через CryptoBot

Після оплати відразу у чат з ботом прийде повідомлення про нарахування коштів. Кошти можна перевірити, якщо перейти в меню (Рисунок 2.14).

Рисунок 2.14 – Повідомлення про успішну оплату та оновлений баланс

Бот дізнався про оплату саме від вебхуку CryptoBot, який самостійно надіслав повідомлення про успішну оплату, а бот передає цю інформацію до користувача.

2.6.4 Сцена початку гри

Сцена початку гри є ключовим етапом у взаємодії користувача з Telegram-ботом, адже саме на цьому кроці відбувається організація ігрової сесії: формування лобі, очікування приєднання інших гравців та запуск самої гри через Telegram MiniApp.

Після натискання кнопки «ПОЧАТИ ГРУ» бот переходить до спеціальної сцени, яка керує усім життєвим циклом ігрового лобі.

Під основним повідомленням Telegram з’являється контекстне меню:

– «Створити лоббі (0.05 USDT)» – ініціює створення нової гри, списуючи відповідну суму з балансу;
– «Приєднатися до лобі» – дозволяє підтвердити приєднання до поточно вибраної сесії;
– «Вийти» – скасовує дію та повертає користувача до головного меню.

Кожне лобі має унікальний ідентифікатор і тимчасово зберігається в Redis з TTL.

Рисунок 2.15 – Сцена початку гри

Після обирання варіанту «Приєднатися до лоббі» з’являється список з усіма доступними лобі, на рисунку 2.15 вже є існуюче лобі №51640, це означає, що в даному лобі вже присутній один учасник із двох необхідних.

Рисунок 2.16 – Посилання на приєднання до гри

Така реалізація забезпечує прозору та зручну взаємодію. Користувач не вводить жодних параметрів вручну – усі дії виконуються через інтуїтивно зрозумілі кнопки. Стан кожного лобі фіксується в Redis, а бот відстежує поточну кількість учасників. У разі, якщо кількість гравців досягла необхідного мінімуму (наприклад, 2/2), система автоматично запускає сесію та надсилає кожному гравцеві персоналізоване посилання на MiniApp-гру (Рисунок 2.16).

2.7 Ігровий процес

Ігрова сесія в Telegram MiniApp грі реалізована на основі вебфронтенду, написаного з використанням React.js, та серверної логіки, яка працює через WebSocket-протокол за допомогою бібліотеки Socket.IO. Така комбінація дозволяє забезпечити двосторонню комунікацію між клієнтом і сервером у режимі реального часу – необхідну умову для синхронного мультиплеєрного ігрового процесу.

Клієнтський застосунок підключається до WebSocket-серверу після запуску гри через Telegram MiniApp. Приєднання відбувається через унікальний ідентифікатор сесії (gameId), який користувач отримує ще в Telegram під час створення або приєднання до лобі.

Рисунок 2.17 – Діаграма співпраці клієнта та сервера

Після підключення:

- клієнт надсилає подію joinGame, що ініціалізує участь у грі;
- сервер повертає startGame з інформацією про поле, гравців і їхні стани;
- клієнт отримує оновлення через події gameUpdate, currentPlayer, diceRolled, chatMessage тощо.

Таким чином, уся взаємодія між гравцями, ігровим полем, логікою покупок, платежів і аукціонів здійснюється за допомогою подієвої моделі Socket.IO.

2.7.1 Правила гри

Гра є цифровою інтерпретацією класичної економічної гри на зразок «Монополії». Її структура складається з таких етапів:

1. Початок гри:

- До початку гри вебсокет чекає усіх гравців, тим, хто вже приєднався демонструється екран завантаження (Рисунок 2.18)
- Після формування повного складу гравців (2 або 4), сервер ініціалізує гру.
- Кожному гравцю надається баланс (200к), позиція (0) та колір фішки.

2. Кидок кубика та переміщення:

- У свій хід гравець надсилає подію rollDice.
- На сервері випадає випадкове число від 1 до 6.
- Гравець пересувається вперед по полю.

3. Типи полів та їх ефекти:

- start – гравець отримує +200 до балансу.
- property – поле можна купити або здавати в оренду іншим гравцям. Доступна побудова філіалів.
- tax – баланс гравця зменшується на фіксовану суму.
- casino – гравець має шанс виграти або програти половину балансу.
- police – гравець потрапляє у "в’язницю" на 2 ходи.
- jail – спеціальне місце, куди телепортується гравець із «поліції».

4. Покупка полів:

- Якщо гравець потрапляє на незайняте поле та має достатньо коштів, система пропонує його купити (promptPurchase).
- Якщо гравець відмовляється або не має коштів – поле виставляється на аукціон (auctionField).

5. Орендна плата:

- Якщо поле належить іншому гравцеві, активний гравець сплачує йому оренду.
- Якщо на полі є філіали – сума збільшується пропорційно.

6. Ігровий чат та обмін:

- Гравці можуть надсилати повідомлення (chatMessage) у внутрішньому чаті.
- Доступний механізм обміну майном та грошима між гравцями (tradeProposal, tradeResponse).

7. Фінансові труднощі:

- Якщо гравець не може оплатити податки або оренду, система автоматично заставляє його майно.
- Якщо навіть заставленого майна недостатньо – гравець вибуває.

8. Кінець гри:

– Якщо залишається лише один гравець – він автоматично оголошується переможцем (endGame).
– Сесія видаляється із серверної пам’яті.
– Усім гравцям надається 10 до experience, та на 15 більше тому, хто переміг, також збільшується кількість зіграних ігор

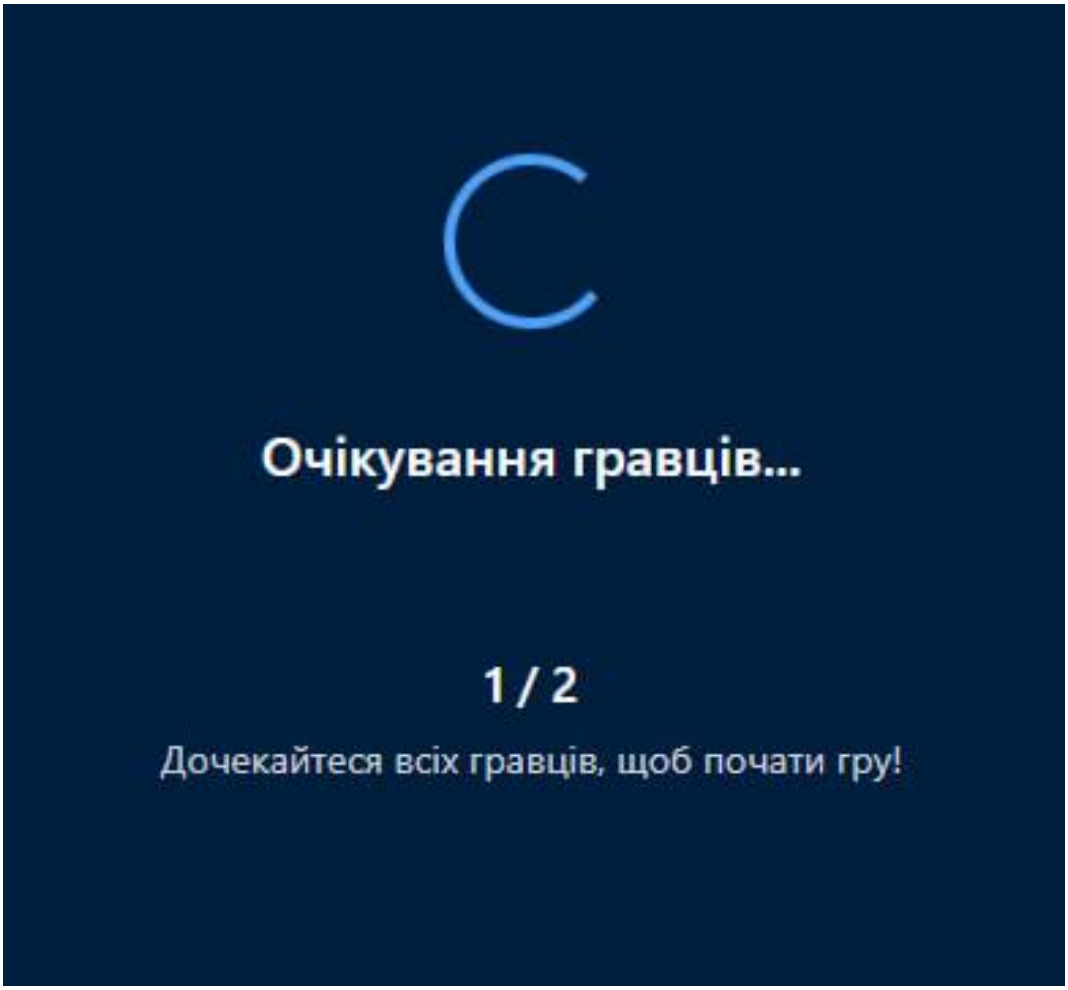

Рисунок 2.18 – Очікування гравців

На початковому етапі кожен гравець бачить екран очікування, де відображається поточна кількість підключених учасників. Як тільки всі необхідні гравці приєднуються до сесії, система автоматично ініціює гру. Такий механізм дозволяє забезпечити синхронний старт для всіх учасників, що є критичним для коректного перебігу ігрового процесу в реальному часі.

2.7.2 Ігрове поле

Ігрове поле – це центральний візуальний та логічний елемент MiniApp-гри. Саме тут відбувається основна динаміка гри: переміщення фішок гравців, купівля активів, сплата податків, побудова філіалів тощо. У межах клієнтського інтерфейсу, реалізованого на через стилі та компоненти, поле відображається у вигляді прямокутного маршруту, що імітує класичну економічну гру за типом «Монополії».

Ігрове поле складається з 16 клітинок, кожна з яких має певний тип і впливає на гравця згідно з попередньо закладеними правилами. Усі клітинки розташовані по периметру ігрового вікна (Рисунок 2.19), а гравці рухаються по колу за годинниковою стрілкою.

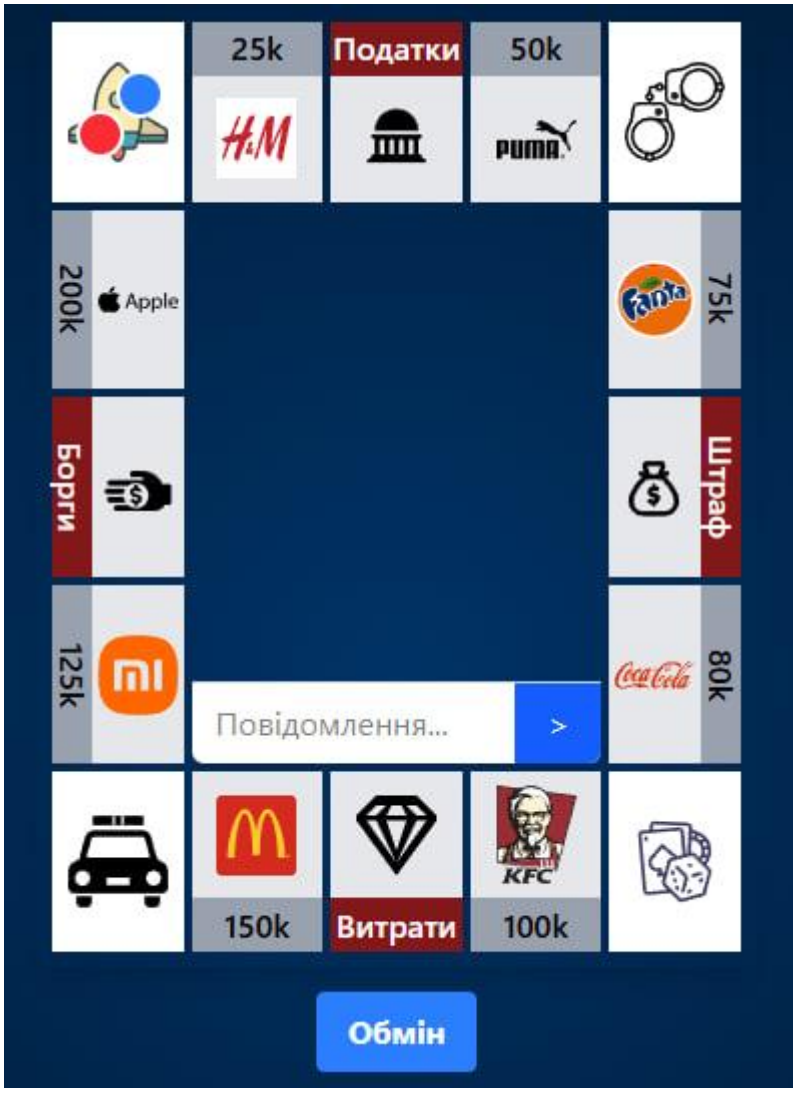

Рисунок 2.19 – Ігрове поле

Додаткові елементи управління

На території поля розташовані інтерфейсні компоненти:

– Поле чату – для надсилання повідомлень іншим учасникам гри;
– Кнопка «Обмін» – ініціація торгової пропозиції;
– Кнопка «Бросити кубик» – основна дія в кожному ході.

У верхній частині інтерфейсу (Рисунок 2.20) виведено блоки активних гравців з їхніми:

– іменами,
– кольором фішки,
– поточним балансом.

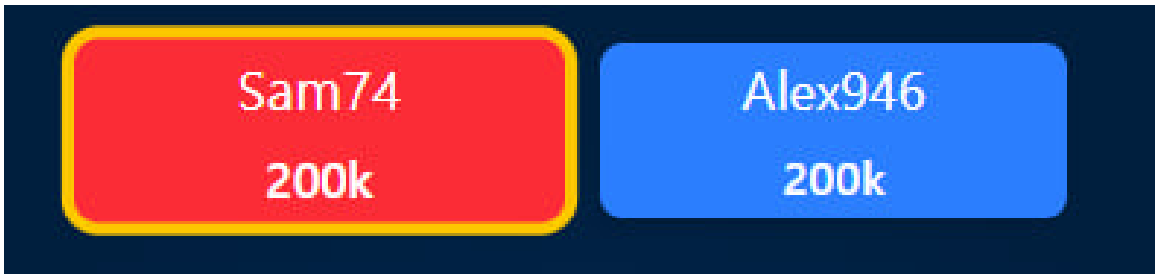

Рисунок 2.20 – Поточні гравці

Кожному гравцю надається унікальний колір (червоний, синій тощо), який використовується як в інтерфейсі, так і в полі. Це дозволяє легко ідентифікувати, хто знаходиться на якій позиції.

Динаміка

Після кожного ходу система автоматично оновлює:

– позиції гравців;
– значення балансу;
– власників полів;
– статус застави або побудованих філіалів;
– черговість гравців.

2.7.3 Обмін майном між гравцями

Один із ключових елементів економічної стратегії гри – обмін активами між гравцями. Він дозволяє гравцям адаптуватися до ситуацій на полі, домовлятися, укладати вигідні угоди та збалансовувати свої ресурси для досягнення переваги.

Функціональність обміну реалізована у вигляді окремого діалогового вікна, яке викликається за допомогою кнопки «Обмін» під час свого ходу. Інтерфейс побудований так, щоб забезпечити максимально просту, але водночас гнучку взаємодію.

Рисунок 2.21 – Модальне вікно обміну

На Рисунок 2.21 зображено приклад відкритого діалогу обміну між гравцями Sam74 і Alex946. Інтерфейс поділено на дві частини:

Верхня частина – визначає, що віддає ініціатор обміну:

- один або кілька об’єктів (нерухомість: Fanta, Coca-Cola);
- бажану суму грошей.

Нижня частина – визначає, що очікує отримати ініціатор від іншого гравця:

- обраного гравця зі списку (у прикладі – Alex946);
- запропоновану нерухомість (у прикладі – McDonalds);
- суму коштів, яку повинен передати цей гравець.

Після заповнення полів користувач натискає «Запропонувати», після чого система надсилає запит другому гравцю, який бачить спливаюче повідомлення з описом пропозиції та кнопками «Прийняти» або «Відхилити».

Взаємодія між клієнтом і сервером реалізована через WebSocket-події:

- tradeProposal – надсилає пропозицію;
- tradeResponse – обробляє прийняття або відхилення;
- tradeResult – інформує ініціатора про результат;
- chatMessage – у разі прийняття або відмови сервер надсилає повідомлення у загальний чат, наприклад:

Гравець Sam74 запропонував гравцю Alex946 поля Fanta, Coca-Cola за McDonalds

Гравець Alex946 відхилив пропозицію

У разі прийняття обміну:

- змінюється власність полів (перепризначення індексу owner);
- коригується баланс кожного гравця;
- відбувається оновлення стану гри (gameUpdate).

2.7.4 Застава та викуп майна

У рамках ігрового процесу реалізовано механіку застави та викупу майна, яка додає грі додаткову глибину та стратегічну варіативність. Вона

дозволяє тимчасово отримати фінансову підтримку шляхом закладення своїх активів, а пізніше – повернути їх у власність через викуп. Така система особливо важлива у випадках, коли гравець стикається з податковими зобов'язаннями, боргами або неспроможністю оплатити оренду чужого майна.

Якщо гравець опиняється в ситуації, коли сума на його балансі недостатня для виконання обов'язкової дії (наприклад, сплати податку або ренти іншому гравцю), гра автоматично запускає процес застави.

Алгоритм працює наступним чином:

1. Визначаються всі не закладені поля, що належать гравцю;
2. Вони сортуються за ціною (від дешевших до дорожчих);
3. По черзі кожне з полів переводиться у стан застави:
   - змінюється колір рамки на сірий (Рисунок 2.22);
   - встановлюється прапорець mortgaged = true;
   - додається лічильник mortgageTurnsLeft = 3 (3 ходи до автоматичного скасування права викупу);
   - сума застави дорівнює повній вартості активу (на відміну від класичної «Монополії», де це половина).

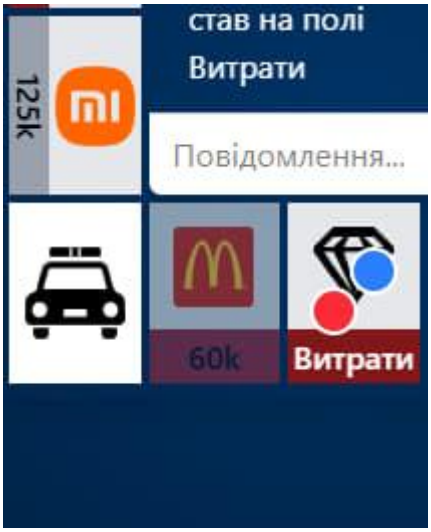

Рисунок 2.22 – Закладене поле

Після завершення кожного ходу, гравець має змогу викупити заставлені об'єкти. Для цього:

1. натискає на відповідне поле у своєму інтерфейсі;
2. у випадаючому меню обирає опцію «Викупити»;
3. якщо баланс достатній – сума списується, а поле повертається у стан активного активу.

Окрім ручного викупу, реалізовано механіку втрати прав на викуп:

- якщо гравець не викупив майно протягом трьох ходів, поле автоматично звільняється;
- воно знову може бути куплене будь-ким.

2.7.5 Завершення гри

Завершення ігрової сесії є логічним підсумком ігрового процесу, який настає у момент, коли в грі залишається лише один активний гравець. Усі інші або збанкрутували (не змогли оплатити зобов’язання навіть після застави), або покинули сесію вручну чи через технічне від’єднання.

Цей механізм виконує декілька важливих функцій:

- офіційно визначає переможця сесії;
- очищує ресурси гри з пам’яті сервера;
- забезпечує гравцю відчуття завершеності ігрового процесу.
- Оновлює його дані на сервері

Ігрова сесія вважається завершеною, якщо:

- у списку гравців залишається лише один учасник (усі інші вибули або від’єдналися);
- гра не перебуває в стані очікування (status === 'playing');
- виконується логіка очищення стану: видаляються дані поля.

На завершення сесії клієнтський застосунок отримує подію endGame, яка активує спливаюче модальне вікно з привітанням переможця. Як видно на Рисунок 2.23, повідомлення оформлено у модальному вікні.

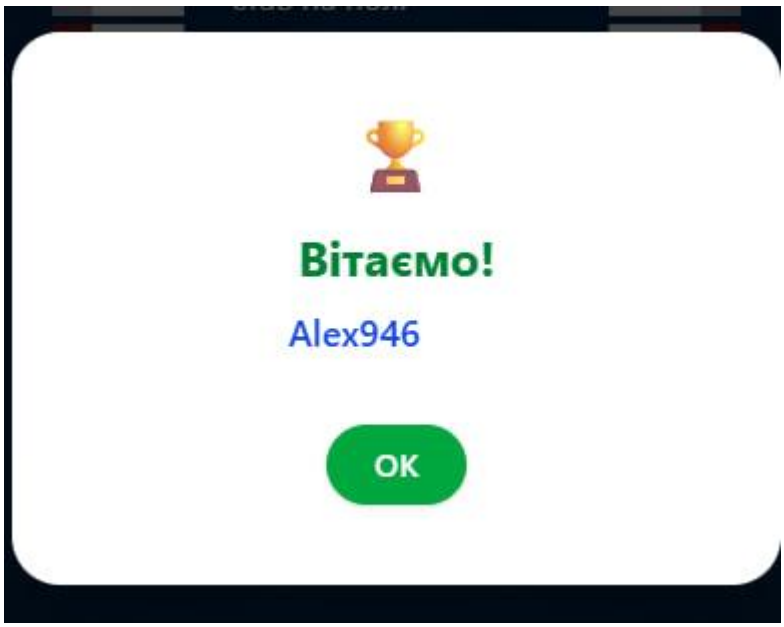

Рисунок 2.23 – Модальне вікно перемоги гравця

Після завершення гри користувач може знову зайти в меню та побачити оновлені дані свого профілю (Рисунок 2.24)

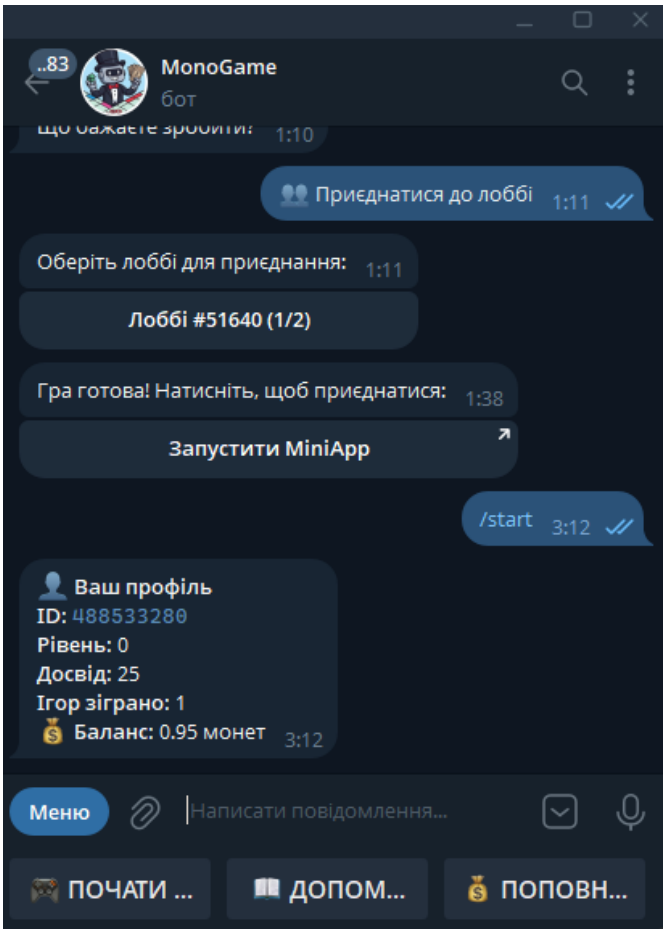

Рисунок 2.24 – Оновлений результат профілю

Був оновлений профіль користувача після завершення однієї ігрової сесії. У порівнянні з початковим станом, відбулися дві ключові зміни:

- Кількість зіграних ігор зросла з 0 до 1, що свідчить про фіксацію результату сесії в системі.
- Баланс користувача зменшився на 0.05 монет (з 1.00 до 0.95), що відповідає вартості створення або участі в лобі.

Висновки до розділу:

- У цьому розділі було сформовано єдиний технологічний стек, що включає Node.js і TypeScript як серверну основу, NestJS як фреймворк, PostgreSQL і Redis для роботи з даними, RabbitMQ для подієвої взаємодії, Prisma ORM для зручної роботи з базою, а також React.js і TailwindCSS на клієнтському боці. WebSocket (Socket.IO) забезпечує обмін даними в реальному часі.
- Розроблено мікросервісну архітектуру, де кожен сервіс відповідає за окрему частину функціоналу – користувачі, баланс, сесії, WebSocket. API Gateway централізує обробку запитів, а RabbitMQ дозволяє досягти асинхронності й масштабованості.
- База даних побудована на PostgreSQL та охоплює основні сутності гри. Структура спроєктована з урахуванням розмежування доступу, використано зовнішні ключі, індекси, тригери. Діаграма БД наочно демонструє логіку зв’язків.
- Створено функціональну модель системи: SADT-діаграма відображає функціональні блоки, а DFD-діаграма – потік даних між користувачем, сервісами та зовнішніми API (Telegram, CryptoBot).
- Пояснено підходи до розгортання проєкту – як у локальному середовищі з використанням ngrok і serveo, так і у хмарній інфраструктурі на базі VPS, Vercel, Fly.io. Docker забезпечує стабільне середовище запуску.

- Особливу увагу приділено Telegram-боту, який є головним інтерфейсом взаємодії. За допомогою сцен Telegraf реалізовано логіку команд, підключення до гри, поповнення балансу, створення лобі тощо.
- Описано повний життєвий цикл гри: підключення до сесії, кидки кубика, пересування, купівля нерухомості, оренда, сплата податків, потрапляння у в’язницю, робота з майном, обміни, застава та викуп активів. Впроваджено механіку завершення гри з визначенням переможця.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано повноцінний застосунок у вигляді Telegram MiniApp гри, що поєднує сучасні принципи веброзробки, мікросервісну архітектуру, платіжні інструменти та гейміфікований підхід до взаємодії з користувачами. Проєкт продемонстрував можливість побудови масштабованого цифрового продукту із залученням лише відкритих технологій і сервісів, доступних кожному розробнику.

Обґрунтований вибір технологічного стеку дозволив створити ефективне, продуктивне та безпечне середовище. Node.js і TypeScript забезпечили надійність та швидкість обробки запитів. Redis використано для кешування і тимчасових станів, а RabbitMQ – для асинхронного обміну між мікросервісами. Використання Docker дозволило забезпечити стабільність, гнучкість та незалежність середовища розробки й розгортання.

Архітектурно система реалізована у вигляді мікросервісної структури, кожен компонент якої відповідає за окрему логічну частину: управління користувачами, балансами, сесіями, платіжною інтеграцією та геймплеєм. Це надало змогу масштабувати, оновлювати та супроводжувати сервіси незалежно один від одного, що підвищує життєстійкість проєкту в умовах зростання навантаження.

На прикладі Telegram-бота було реалізовано зручну модель взаємодії з користувачем. Через використання сцен та сесійної пам’яті користувацький досвід став гнучким та інтуїтивним. Бот обробляє всі команди, супроводжує користувача на всіх етапах, дозволяє поповнювати баланс, створювати ігри, приєднуватися до лобі, а також ініціювати ігрову сесію. Всі дії виконуються без необхідності покидати середовище Telegram, що значно знижує поріг входу для гравця.

Ключовою частиною проєкту став ігровий модуль – цифрова реалізація покрокової настільної гри за зразком «Монополії», реалізована на React.js з обміном через WebSocket. Забезпечено повноцінну інтерактивність у режимі

реального часу між клієнтом і сервером: рух фішок, оплати, покупка майна, оренда, торги, застава, банкрутство, перемога. Кожна сесія є унікальною і зберігається окремо, що дозволяє відслідковувати результати та досвід гравців.

Запровадження криптовалютної платіжної системи на базі CryptoBot продемонструвало можливість реалізації безпечних фінансових операцій без складної інтеграції з банківськими сервісами. Користувач отримує рахунок, проводить оплату через WebApp, а підтвердження надходить автоматично через Webhook.

Таким чином, отриманий результат може бути використаний як основа для подальшої розробки повноцінного ігрового сервісу з підтримкою масштабних турнірів, маркетплейсу цифрових активів та іншого.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Крамер Н., 10 найкращих практик мікросервісів Node.js 2024. daily.dev. URL: https://daily.dev/blog/10-nodejs-microservices-best-practices-2024.

2. Як використовувати з React | Socket.IO. Socket.IO. URL: https://socket.io/how-to/use-with-react.

3. Інформаційні технології – 2025: зб. тез XII Всеукраїнської науковопрактичної конференції молодих учених, 15 трав. 2025 р., м. Київ / Київський столичний університет імені Бориса Грінченка; Відповід. за вип.: М.М. Астаф'єва, Д.М. Бодненко, О.М. Глушак, І.В. Машкіна, О.В. Локазюк, В.В. Прошкін, С.М. Шевченко. Київ : Київський столичний університет імені Бориса Грінченка, 2025. 362 с. ISSN: 2664-2638.

4. Офіційний сайт компанії RabbitMQ. rabbitmq.com. URL: https://www.rabbitmq.com.

5. Яскевич В. JavaScript бібліотека React. Навчальний посібник для студентів спеціальності Комп'ютерні науки. Київ : Київ. ун-т ім. Бориса Грінченка, 2023. 63 с.

6. Мамбу Р. ЯК НАЛАШТУВАТИ МІКРОСЕРВІС З NESTJS, RABBITMQ, TYPEORM ТА POSTGRES. Medium. URL: https://medium.com/@ryanmambou/how-to-setup-a-microservice-with-nestjs-rabbitmq-typeorm-and-postgres-2eb691c17e17.

7. Бази даних та засоби управління. Практикум. [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 – Комп'ютерна інженерія. / В.І. Павловський, А.В. Петрашенко, Д.В. Победа; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 7,7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 112 с.

8. Уроки від W3Schools українською онлайн. W3Schools українською. URL: https://w3schoolsua.github.io/js/index.html.

9. Redis. Офіційна сторінка Nest.JS. URL: https://docs.nestjs.com/microservices/redis.

10. TailwindCSS. Офіційна документація TailwindCSS. URL: https://tailwindcss.com/docs/.

11. Юсеф М. Н. Створення масштабованих мікросервісів за допомогою NestJS: практичний посібник. Medium. URL: https://medium.com/@nafihpp/building-scalable-microservices-with-nestjs-a-practical-guide-fa531f5b4f4c.

12. Санаткар Х. Telegraf JS: Бібліотека для створення телеграм-бота за допомогою Javascript. DEV Community. URL: https://dev.to/hoomehrsanatkar/telegrafjs-library-for-creating-telegram-bot-with-javascript-ek6.

13. Telegraf.js. Офіційна документація telegraf.js. URL: https://telegraf.js.org/.

14. Шарма Р. Від коду до чату: створення бота Telegram за допомогою Node.js і Telegraf. Medium. URL: https://medium.com/@ravisharma23523/from-code-to-chat-creating-a-telegram-bot-with-node-js-and-telegraf-69f096e31364.

15. Як грати в «Монополію»: правила, види, прийоми і цікаві факти - Neuron Toys. Neuron Toys. URL: https://neuron-toys.com/uk/kak-igrat-v-monopoliyu-pravila-vidy-priemy-i-interesnye-fakty.

16. Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с. (для спеціальності КН)

17. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

18. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

19. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

20. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).