

Технічне завдання (ТЗ) на розробку платформи з картками, аукціонами та блокчейном

1. Загальний опис проєкту

1.1. Суть гри

Гравці можуть зареєструватися на платформі, після чого їм доступні наступні механіки:

- Лотерея – раз на певний період гравець може отримати випадкову картку.
- Колекціонування – зібрані картки зберігаються у профілі гравця.
- Торгівля – можливість продавати/купувати картки на аукціоні.
- Ігрова валюта – гравці отримують внутрішні гроші за активність (вхід у гру, запрошення друзів, продаж карток).
- Блокчейн – всі картки є NFT та закріплюються за власником у блокчейні.

1.2. Основні користувачі

- Гравець – реєструється, отримує картки, бере участь у торгах.
- Адміністратор – модерування аукціонів, управління базою карток.

1.3. Основні фічі

- ✓ Генерація карток з різними рідкісностями
 - ✓ Аукціони та торги між гравцями
 - ✓ Блокчейн для збереження власності карток
 - ✓ Внутрішня валюта та механізм отримання бонусів
 - ✓ Гнучка система API для масштабування
-

2. Архітектура проєкту

2.1. Загальна схема

Проєкт розробляється у мікросервісній архітектурі.

Основні мікросервіси:

- **service-user** – реєстрація, авторизація, профілі
- **service-card** – управління картками
- **service-lottery** – механіка отримання карток
- **service-trade** – продаж та аукціони
- **service-wallet** – управління ігровою валютою
- **service-blockchain** – взаємодія з блокчейном
- **service-gateway** – єдина точка входу (API Gateway)

2.2. Взаємодія сервісів

Взаємодія між сервісами реалізована через:

- gRPC – для швидкої синхронної комунікації
 - RabbitMQ – для обміну подіями між сервісами
 - REST API – для зовнішнього клієнтського інтерфейсу
-

3. Технологічний стек

3.1. Backend

- ✓ Node.js + NestJS – для розробки мікросервісів
- ✓ TypeScript – для типізації
- ✓ PostgreSQL / MongoDB – зберігання користувачів, карток, історії
- ✓ Redis – кешування
- ✓ RabbitMQ – брокер повідомлень для обміну між сервісами
- ✓ gRPC / REST API – для взаємодії між сервісами

3.2. Blockchain

- ✓ Ethereum / Polygon – для збереження NFT-карток
- ✓ Smart Contracts (Solidity) – смарт-контракти для торгівлі

3.3. Frontend

- ✓ React + Next.js – для веб-інтерфейсу
- ✓ TailwindCSS – для стилізації

3.4. DevOps

- ✓ Docker + Kubernetes – деплой всіх мікросервісів
 - ✓ CI/CD (GitHub Actions / GitLab CI) – автоматизація деплою
-

4. База даних та основні сутності

4.1. Основні таблиці та їх атрибути

1 Користувачі (**users**)

- **id** – унікальний ідентифікатор
- **email** – електронна пошта
- **password** – хеш пароля
- **balance** – баланс внутрішньої валюти
- **wallet_address** – прив'язка до блокчейну

2 Картки (**cards**)

- **id** – унікальний ідентифікатор
- **name** – назва
- **rarity** – рідкість (common, rare, epic, legendary)
- **image_url** – посилання на зображення
- **owner_id** – ID власника

3 Аукціони (**auctions**)

- **id** – унікальний ідентифікатор
- **card_id** – картка, що продається
- **seller_id** – продавець
- **start_price** – стартова ціна
- **end_time** – час закінчення торгів

4 Транзакції (**transactions**)

- **id** – унікальний ідентифікатор
- **user_id** – відправник
- **receiver_id** – отримувач
- **amount** – сума
- **type** – тип (лотерея, покупка, продаж)

5. Бізнес-логіка мікросервісів

5.1. Алгоритм отримання картки

1. Гравець запускає лотерею в `service-lottery`
2. Система визначає випадкову рідкість картки
3. Картка створюється у `service-card`
4. Власник записується у blockchain через `service-blockchain`
5. Картка з'являється у профілі гравця

5.2. Алгоритм продажу картки

1. Гравець додає картку на аукціон у `service-trade`
 2. Інші користувачі можуть робити ставки
 3. Коли аукціон завершується:
 - Гроші переказуються через `service-wallet`
 - Власність змінюється у blockchain
 - Картка переходить до нового власника
-

6. Безпека

- Використання JWT для аутентифікації
 - Шифрування паролів bcrypt
 - Захист API за допомогою rate limiting
 - Захист фінансових операцій через 2FA
-

7. Масштабування

- ◆ Горизонтальне масштабування мікросервісів через Kubernetes
 - ◆ Використання Redis для кешування запитів
 - ◆ CDN (Cloudflare / AWS CloudFront) для оптимізації завантаження карток
-

8. Очікувані результати

- ✓ Запуск MVP-версії гри з базовими механіками
- ✓ Інтеграція блокчейну для управління картками
- ✓ Запуск торгового майданчика для карток

9. Моніторинг та логування

9.1. Моніторинг мікросервісів

Щоб відстежувати стан сервісів і швидко реагувати на проблеми, використовуємо:

- Prometheus – збір метрик (нагрузка CPU, RAM, час відповіді сервісів)
- Grafana – візуалізація метрик та алерти
- Kubernetes Health Checks – перевірка живучості сервісів (*livenessProbe*, *readinessProbe*)

Основні метрики:

- ✓ Запити до API (кількість, успішні/помилки)
- ✓ Час відповіді сервісів (*response time*)
- ✓ Кількість активних користувачів
- ✓ Використання ресурсів (CPU, RAM, дисковий простір)
- ✓ Кількість подій у RabbitMQ (чи не зависла черга)
- ✓ Статистика транзакцій та аукціонів

9.2. Логування

Для логування подій та помилок використовуємо Winston + Elasticsearch:

- Winston – централізоване логування у всіх мікросервісах
- Elasticsearch + Kibana – збір логів та їх перегляд у реальному часі




Типи логів:

- Інформаційні (INFO) – запити користувачів, зміни у профілях
- Попередження (WARNING) – затримки в обробці запитів, нестача ресурсів
- Помилки (ERROR) – збої сервісів, невдалі транзакції, критичні баги

Приклад формату логів (json):

```
{  
  "timestamp": "2024-02-01T12:34:56Z",  
  "level": "error",  
  "service": "service-trade",  
  "message": "Аукціон не завершився через помилку бази даних",  
  "context": {  
    "auctionId": "12345",  
    "sellerId": "67890"  
  }  
}
```

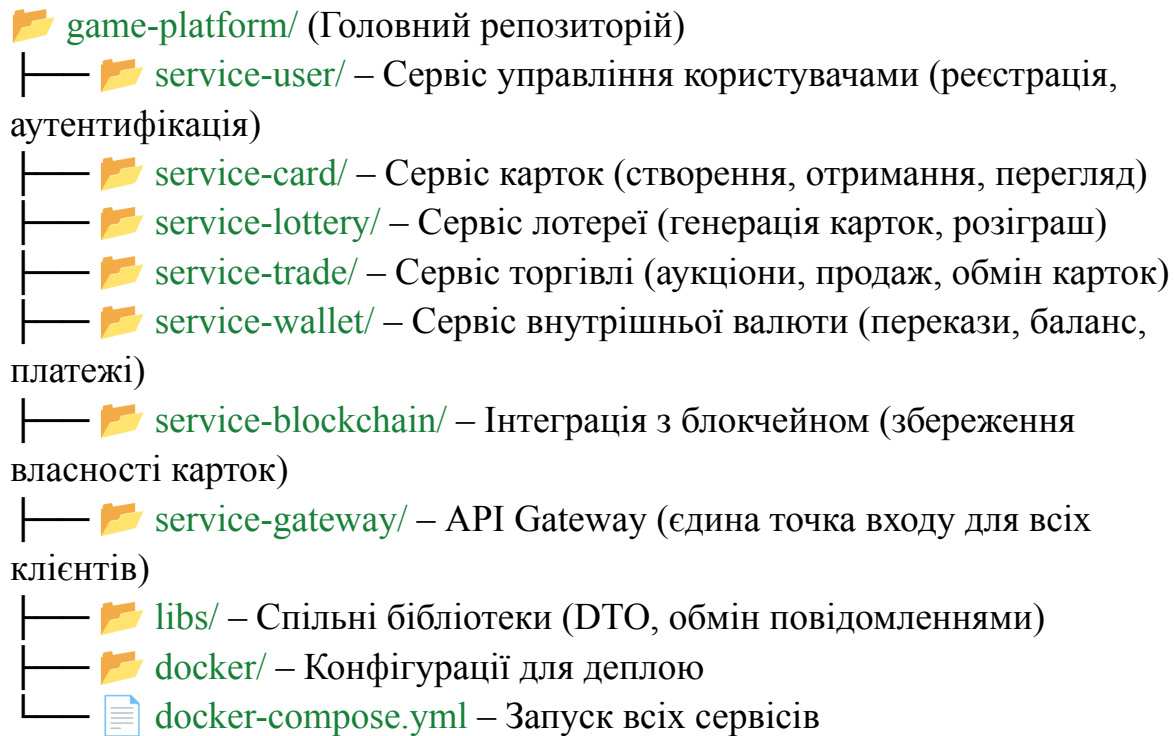
9.3. Сповіщення про критичні помилки

-  Slack / Telegram боти – надсилення повідомлень у канал розробників
 -  Email-сповіщення – якщо сервіс падає більше ніж на 5 хвилин
 -  Webhook-інтеграції – підключення до PagerDuty для швидкого реагування
-

9.4. Автоматичне відновлення сервісів

- ⚙️ Kubernetes Auto Healing – якщо сервіс впав, Kubernetes його перезапустить
- ⚙️ Rate Limiting + Circuit Breaker – захист від перевантаження та DDoS

Пропонована мікросервісна структура:



Коротко про мікросервіси:

1 service-user

- Реєстрація, вхід, управління профілем
- Використовує **JWT** для аутентифікації
- Зберігає користувачів у базі (MongoDB/PostgreSQL)

2 service-card

- Генерація карток
- Отримання списку карток користувача
- Взаємодія з **service-blockchain** для власності

3 service-lottery

- Механізм отримання карток через лотерею
- Обчислення рідкості картки
- Підключення до **service-card**

4 service-trade

- Додавання картки на аукціон
- Купівля/продаж карток між гравцями
- Використовує **service-wallet** для платежів

5 service-wallet

- Управління внутрішньою валютою
- Баланс користувача
- Платежі за картки, бонуси за вхід

6 service-blockchain

- Збереження власності карток на блокчейні
- Перевірка транзакцій

7 service-gateway

- **API Gateway** для всіх запитів
 - Глобальна аутентифікація
-

Взаємодія мікросервісів:

- ✓ Використовуємо **RabbitMQ** для асинхронного обміну подіями
- ✓ Всі сервіси працюють через **gRPC/HTTP**
- ✓ Деплой в **Docker + Kubernetes**