

# **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**

Проект розробляється у мікросервісній архітектурі. Основні сервіси:

#### **1. API Gateway**

##### **Мета:**

Єдина точка входу для всіх зовнішніх запитів (від веб-клієнтів, мобільних додатків тощо), яка забезпечує маршрутизацію, автентифікацію, авторизацію та базове логування.

##### **Основні функції та особливості:**

- **Маршрутизація:**

Розподіляє запити до відповідних сервісів (User & Auth, service-core, Transaction & Smart Contract Integration, Notification).

- **Безпека:**

Виконує первинну автентифікацію та авторизацію (наприклад, перевірку JWT), що дозволяє іншим сервісам зосередитися на бізнес-логіці.

- **Агрегація відповідей:**

Може об'єднувати відповіді з кількох сервісів у разі потреби.

- **Контроль навантаження:**

Реалізує механізми обмеження швидкості (rate-limiting) та захист від DDoS-атак.

##### **Взаємодія:**

- Приймає запити від клієнтів та далі передає їх до інших сервісів через REST/GraphQL API або інші протоколи.

- Забезпечує єдиний інтерфейс для всієї системи, ізолюючи внутрішню логіку від зовнішнього світу.
- 

## **2. User Service**

### **Мета:**

Управління користувачами, автентифікацією, авторизацією та зберіганням профільних даних.

### **Основні функції та особливості:**

- **Реєстрація та вхід:**

Забезпечує створення облікових записів, валідацію даних, хешування паролів та видачу токенів (наприклад, JWT або OAuth2).

- **Управління профілем:**

Зберігає інформацію про користувача, таку як email, ім'я, wallet\_address, баланс внутрішньої валюти тощо.

- **Сесійний менеджмент:**

Відповідає за підтримку активних сесій, оновлення токенів та управління правами доступу.

- **Безпека:**

Інтегрується з API Gateway для перевірки автентифікації; забезпечує централізоване зберігання даних користувачів з високим рівнем захисту.

### **Взаємодія:**

- Викликається API Gateway для первинної перевірки запитів.
- Інші сервіси (наприклад, Transaction & Smart Contract Integration Service) отримують від нього актуальну інформацію про користувачів (баланси, права доступу) через API.

- Не має прямого доступу до даних з service-core, але може використовувати внутрішні API для потреб перевірки прав доступу при здійсненні транзакцій.
- 

### 3. Service-Core

#### Мета:

Централізований сервіс для управління всіма операціями з колекційними картками, аукціонами, ставками та пов'язаними даними, з доступом до основної бази даних.

#### Основні функції та особливості:

- **Управління картками:**

CRUD-операції над колекційними картками (створення, оновлення, видалення, отримання даних).

Зберігає деталі картки: назва, рідкість, зображення, поточний власник, статус (наприклад, *available*, *in\_auction*, *sold*).

- **Аукціони та ставки:**

- Створення аукціонів для карток (за умовами – встановлення стартової ціни, часових рамок, тощо).
- Обробка ставок. Включає логіку перевірки мінімального підвищення, валідацію балансу користувача та запис історії ставок.
- Визначення переможця після завершення аукціону.

- **Централізований доступ до даних:**

Єдиний інтерфейс для доступу до основної бази даних, що дозволяє іншим сервісам отримувати дані через API, не маючи прямого доступу до бази.

- **Внутрішні події:**

Публікує події (наприклад, *auction\_created*, *new\_bid*, *auction\_closed*) через Event Bus, що дозволяє іншим сервісам реагувати на зміни.

**Взаємодія:**

- API Gateway надсилає запити для операцій із картками та аукціонами.
  - Інші сервіси (наприклад, Transaction & Smart Contract Integration Service) отримують від нього інформацію про завершені аукціони чи зміни власності.
  - Не надає прямого доступу до бази даних іншим сервісам, гарантуючи централізований контроль над даними.
- 

#### **4. Transaction & Smart Contract Integration Service**

**Мета:**

Обробка фінансових операцій, транзакцій між користувачами, внутрішніх переказів, а також інтеграція зі смарт-контрактами для забезпечення прозорості та безпеки критичних операцій.

**Основні функції та особливості:**

- **Фінансові транзакції:**
  - Обробка операцій покупки/продажу, лотерей та внутрішніх переказів.
  - Зберігання історії фінансових операцій для аудиту.
  - Взаємодія з User & Auth Service для оновлення балансу користувача.
- **Інтеграція зі смарт-контрактами:**

- Виклик смарт-контрактів для автоматичного переведення власності картки після завершення аукціону.
- Виконання ескроу-операцій: блокування коштів до підтвердження умов угоди.
- Токенізація внутрішньої валюти: управління створенням, переказом і спалюванням токенів.
- **Логіка обробки подій:**

Реагує на події, опубліковані **service-core** (наприклад, завершення аукціону) через Event Bus, ініціюючи відповідні фінансові операції.

#### **Взаємодія:**

- Отримує події від service-core про завершення аукціонів та зміни власності.
  - Звертається до зовнішніх blockchain API/SDK для взаємодії зі смарт-контрактами.
  - Співпрацює з User & Auth Service для оновлення фінансових даних користувачів.
  - Забезпечує, щоб фінансові операції були незалежними від основної бази даних, але отримували потрібні дані через API з service-core.
- 

## **5. Notification Service**

#### **Мета:**

Централізовано інформувати користувачів про важливі події в системі через email, push-повідомлення, SMS тощо.

#### **Основні функції та особливості:**

- **Обробка подій:**

Підписується на події з Event Bus, отримуючи інформацію про зміни в аукціонах, транзакціях, ставках тощо.

- **Надсилання повідомлень:**

Формує та відправляє повідомлення користувачам про ключові події, такі як:

- Створення або завершення аукціону.
- Надходження нової ставки.
- Підтвердження транзакцій або зміни балансу.

- **Гнучкість повідомлень:**

Забезпечує підтримку різних каналів повідомлень та шаблонів, що дозволяє адаптувати систему до різних сценаріїв сповіщення.

## **Взаємодія:**

- Отримує події від інших сервісів (публікуються через Event Bus).
- Не має прямого доступу до основних бізнес-даних, а отримує інформацію через публікацію подій або API Gateway, якщо потрібно здійснити окремі запити.
- Забезпечує асинхронне інформування користувачів без навантаження основних сервісів.

---

## **Інфраструктурні Компоненти**

### **Event Bus / Message Broker:**

- **Мета:** Забезпечення асинхронної комунікації між сервісами через публікацію та підписку на події.
- **Особливості:**

- Використовується для сповіщення Notification Service, для передачі подій від service-core до Transaction & Smart Contract Integration Service та інших.
- Інтеграція з такими інструментами, як RabbitMQ, Apache Kafka або іншими рішеннями.

### **Service Discovery та Configuration Management:**

- **Мета:** Автоматичне виявлення сервісів у кластері (наприклад, Kubernetes, Consul) та централізоване управління конфігураціями.
  - **Особливості:**
    - Забезпечує стабільність та узгодженість параметрів серед усіх сервісів.
    - Допомогає у масштабуванні та моніторингу роботи системи.
- 

### **Загальна Взаємодія Сервісів**

#### **1. Запит від користувача:**

Користувач робить запит через API Gateway, наприклад, для створення аукціону або розміщення ставки.

#### **2. User & Auth Service:**

Аутентифікує користувача та видає токен, перевіряє права доступу.

#### **3. Service-Core:**

Обробляє запити, пов'язані з колекційними картками та аукціонами, оновлює базу даних і публікує події (наприклад, завершення аукціону, зміна статусу картки) через Event Bus.

#### **4. Transaction & Smart Contract Integration Service:**

Отримує повідомлення про завершення аукціону, ініціює фінансові операції, звертається до смарт-контрактів для переведення власності та виконує ескроу-операції.



## 5. Notification Service:

Підписується на відповідні події та надсилає повідомлення користувачам про зміни у статусі аукціонів, ставках або фінансових транзакціях.

---

Цей підхід із централізованим доступом до основної бази даних через **service-core** дозволяє зберегти єдиний канал для операцій з даними, знижуючи складність інтеграції інших сервісів. Кожен сервіс виконує свою чітко визначену роль:

- **API Gateway:** єдиний вхід для зовнішніх запитів та базова безпека.
- **User & Auth Service:** управління користувачами та сесіями.
- **Service-Core:** управління картками, аукціонами та ставками із централізованим доступом до бази.
- **Transaction & Smart Contract Integration Service:** обробка фінансових операцій та інтеграція зі смарт-контрактами.
- **Notification Service:** централізоване інформування користувачів.

Інфраструктурні компоненти (Event Bus, Service Discovery) підтримують асинхронну взаємодію та забезпечують масштабованість і стабільність системи. Ця архітектура забезпечує баланс між модульністю, безпекою та керованістю, що особливо важливо для систем, інтегрованих із блокчейном та смарт-контрактами.

## 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. Таблиця Users (Користувачі)

Призначення: Зберігання даних про користувачів.

Поля:

- **user\_id** (PK, INT, AUTO\_INCREMENT) – унікальний ідентифікатор користувача.
- **email** (VARCHAR) – електронна пошта (унікальне значення).
- **password\_hash** (VARCHAR) – хеш пароля.
- **wallet\_address** (VARCHAR) – адреса гаманця в блокчейні.
- **balance** (DECIMAL) – баланс внутрішньої валюти.
- **is\_active** (BOOLEAN) – аккаунт активний/неактивний.
- **failed\_login\_attempts** (SMALLINT) – число невдалих логінів.
- **created\_at** (DATETIME) – час створення запису.
- **updated\_at** (DATETIME) – час останньої зміни запису.
- **locked\_date** (DATETIME) – час деактивування аккаунту.
- **last\_login\_date** (DATETIME) – час останнього входу в аккаунт.

---

#### 2. Таблиця Cards (Картки)

Призначення: Зберігання інформації про картки, їх характеристик та поточного стану.

Поля:

- **card\_id** (PK, INT, AUTO\_INCREMENT) – унікальний ідентифікатор картки.
- **name** (VARCHAR) – назва картки.
- **rarity** (ENUM або посилання на таблицю **Rarities**) – рідкість (наприклад: 'common', 'rare', 'epic', 'legendary').
- **image\_path** (VARCHAR) – посилання на зображення картки.
- **owner\_id** (INT, FK → Users.user\_id) – поточний власник картки.
- **status** (ENUM: 'available', 'in\_auction', 'sold') – статус картки.
- **created\_at** (DATETIME)
- **updated\_at** (DATETIME)

*Примітка:* Якщо можливе розширення набору значень для рідкості, можна створити окрему таблицю Rarities.

---

### 3. Таблиця Auctions (Аукціони)

Призначення: Організація аукціонів для продажу карток.

Поля:

- **auction\_id** (PK, INT, AUTO\_INCREMENT) – унікальний ідентифікатор аукціону.
- **card\_id** (INT, FK → Cards.card\_id, UNIQUE) – картка, що виставляється на аукціон.

*Зауваження:* у картки, що перебувають на аукціоні, статус оновлюється до **in\_auction**.

- **seller\_id** (INT, FK → Users.user\_id) – продавець (власник картки під час виставлення).
- **start\_price** (DECIMAL) – стартова ціна.

- **current\_price** (DECIMAL) – поточна найвища ставка (може оновлюватися автоматично).
- **start\_time** (DATETIME) – час початку аукціону.
- **end\_time** (DATETIME) – час завершення аукціону.
- **status** (ENUM: 'active', 'closed', 'cancelled') – статус аукціону.
- **winner\_id** (INT, FK → Users.user\_id, NULLABLE) – переможець аукціону (визначається після завершення).
- **created\_at** (DATETIME)
- **updated\_at** (DATETIME)

*Примітка:* Зберігання поточного стану аукціону дозволяє швидко отримувати інформацію про активні торги.

#### 4. Таблиця Bids (Ставки)

Призначення: Збереження історії ставок для кожного аукціону.

Поля:

- **bid\_id** (PK, INT, AUTO\_INCREMENT) – унікальний ідентифікатор ставки.
- **auction\_id** (INT, FK → Auctions.auction\_id) – посилання на аукціон.
- **bidder\_id** (INT, FK → Users.user\_id) – користувач, який зробив ставку.
- **bid\_amount** (DECIMAL) – сума ставки.
- **bid\_time** (DATETIME) – час розміщення ставки.

*Примітка:* Такий підхід дозволяє зберігати повну історію торгів і забезпечує прозорість процесу.

#### 5. Таблиця Wallet\_Transactions (Транзакції)

Призначення: Фіксація фінансових операцій, пов'язаних із системою (покупки, продажі, лотереї, внутрішні перекази тощо).

Поля:

- **transaction\_id** (PK, INT, AUTO\_INCREMENT) – унікальний ідентифікатор транзакції.
- **sender\_id** (INT, FK → Users.user\_id, NULLABLE) – відправник коштів (може бути NULL для надходжень, наприклад, від системи чи лотереї).
- **receiver\_id** (INT, FK → Users.user\_id, NULLABLE) – отримувач коштів.
- **amount** (DECIMAL) – сума транзакції.
- **type** (ENUM: 'purchase', 'sale', 'lottery', 'transfer') – тип транзакції.
- **related\_auction\_id** (INT, FK → Auctions.auction\_id, NULLABLE) – посилання на аукціон (якщо транзакція пов'язана з торгами).
- **related\_card\_id** (INT, FK → Cards.card\_id, NULLABLE) – посилання на картку (за необхідності).
- **created\_at** (DATETIME)

*Примітка:* Поля **sender\_id** та **receiver\_id** дозволяють гнучко обробляти різні типи операцій.

Додаткові зауваження та логіка взаємодії

1. Зв'язки та цілісність даних:

- Для всіх зовнішніх ключів слід задати відповідні обмеження (foreign key constraints), що допомагає підтримувати цілісність даних.
- У випадку, коли картка виставляється на аукціон, необхідно оновити її статус (наприклад, змінити з **available** на **in\_auction**), а після завершення аукціону – відповідно, змінити власника та статус на **sold** або повернути статус у випадку скасування.

2. Історія ставок:

- Таблиця **Bids** дозволяє зберігати повну історію торгів, що є важливим для аудиту та визначення переможця аукціону.
- **current\_price** в таблиці **Auctions** може оновлюватися при кожній новій ставці.

### 3. Транзакції:

- Всі фінансові операції (наприклад, зарахування коштів від продажу, списання при покупці, участь у лотереї) фіксуються у таблиці **Wallet\_Transactions**.
- Поля **related\_auction\_id** та **related\_card\_id** допомагають встановити зв'язок між транзакцією і конкретним аукціоном чи картою, що може бути корисно для розширеної аналітики.

### 4. Модульність:

- При необхідності можна додати додаткові таблиці (наприклад, **Rarities** для гнучкого управління категоріями рідкості або таблицю **Wallet\_Transactions** для детального обліку операцій із блокчейном).

```

1  Users
2    └─ user_id (PK)
3    └─ email, password_hash, wallet_address, balance, created_at, updated_at
4
5  Cards
6    └─ card_id (PK)
7    └─ name, rarity, image_url, owner_id (FK → Users.user_id), status, created_at, updated_at
8
9  Auctions
10   └─ auction_id (PK)
11   └─ card_id (FK → Cards.card_id, UNIQUE)
12   └─ seller_id (FK → Users.user_id)
13   └─ start_price, current_price, start_time, end_time, status, winner_id (FK → Users.user_id), created_at, updated_at
14
15  Bids
16   └─ bid_id (PK)
17   └─ auction_id (FK → Auctions.auction_id)
18   └─ bidder_id (FK → Users.user_id)
19   └─ bid_amount, bid_time
20
21  Transactions
22   └─ transaction_id (PK)
23   └─ sender_id (FK → Users.user_id), receiver_id (FK → Users.user_id)
24   └─ amount, type, related_auction_id (FK → Auctions.auction_id), related_card_id (FK → Cards.card_id), created_at
25

```

## 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

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

### **10. Smart-контракти**

Smart-контракти можуть відігравати ключову роль у системі, забезпечуючи автоматизацію, прозорість і безпеку операцій, що виконуються на блокчейні. Нижче наведено кілька варіантів, де та як можна інтегрувати smart-контракти в описану систему:

---

#### **1. Управління власністю та транзакціями**

- **Переведення власності:**

При успішному завершенні аукціону smart-контракт може автоматично перевести власність на картку від продавця до переможця. Це забезпечує довіру між учасниками, оскільки умови угоди закріплені в незмінному коді smart-контракту.

- **Ескроу-сервіс:**

Smart-контракт може виступати як ескроу (депозитарій), утримуючи кошти покупця до завершення угоди. Після підтвердження успішної передачі картки контракт автоматично передасть кошти продавцю. Це допомагає знизити ризики шахрайства.

---

#### **2. Децентралізовані аукціони**

- Автоматизація аукціонного процесу:

Весь процес аукціону, включаючи прийом ставок, оновлення поточної ставки та визначення переможця, може бути реалізований у smart-контракті. Такий підхід забезпечує прозорість: кожен учасник може перевірити всі транзакції на блокчейні.

- Динамічне оновлення ставок:

Smart-контракт може автоматично обробляти ставки, перевіряти їх відповідність мінімальним критеріям (наприклад, мінімальне підвищення ставки) та оновлювати стан аукціону без участі центрального сервера.

---

### 3. Взаємодія з внутрішньою валютою

- Токенізація балансу:

Якщо ваша система використовує внутрішню валюту, її можна представити у вигляді токенів на блокчейні. Smart-контракт управляє створенням, обігом та спалюванням токенів. Це забезпечує прозорість фінансових операцій, оскільки всі транзакції токенів записуються в публічному реєстрі.

- Платіжні операції:

Smart-контракти можуть здійснювати автоматичні платежі між користувачами при покупці, продажі або участі у лотереї. Це гарантує, що кошти переміщуються лише за виконанням визначених умов.

---

### 4. Інтеграція з базою даних

- Гібридна модель:

Основні дані (користувачі, картки, аукціони, ставки) зберігаються у централізованій базі даних для забезпечення швидкого доступу та зручного інтерфейсу. При цьому фінансові операції та зміна власності карток, які вимагають високої довіри, реалізуються через smart-контракти.

- Верифікація транзакцій:

База даних може містити посилання на транзакції, які зафіксовані в блокчейні (наприклад, ідентифікатор транзакції smart-контракту). Це дозволяє перевіряти автентичність і стан угод.

---

## 5. Приклади сценаріїв використання

- Аукціон картки:

1. Продавець виставляє картку на аукціон через інтерфейс, і запис створюється в базі даних з відповідними деталями.
2. Smart-контракт створюється або викликається, реєструючи аукціон на блокчейні.
3. Учасники роблять ставки через інтерфейс, при цьому кожна ставка реєструється як транзакція в smart-контракті.
4. По завершенню аукціону smart-контракт автоматично визначає переможця, переводить власність картки та здійснює переказ коштів.

- Переказ внутрішніх токенів:

1. Користувач хоче поповнити баланс, надсилаючи токени на свій гаманець.
2. Smart-контракт фіксує операцію поповнення, після чого база даних оновлює значення балансу користувача на основі підтвердженої транзакції в блокчейні.

Smart-контракти використовуються для критичних операцій, де потрібна максимальна прозорість, автоматизація та безпека — наприклад, переведення власності, проведення аукціонів, здійснення платежів і управління токенами. В інтегрованій системі база даних та smart-контракти можуть співпрацювати таким чином, що база даних зберігає інформацію для зручного доступу і аналітики, а smart-контракти гарантують децентралізоване виконання умов угод та незмінність записів.

## 11. Архітектурний підхід

### service-gateway

- Єдина точка входу для всіх зовнішніх запитів. Забезпечує базову автентифікацію, авторизацію та маршрутизацію запитів до внутрішніх сервісів.

#### Основні особливості:

- **Маршрутизація:** Передає запити до відповідних сервісів (User & Auth, Service-Core, Transaction, Notification).
- **Безпека:** Виконує глобальну аутентифікацію (наприклад, перевірку JWT) і застосовує політики доступу.
- **Агрегація:** За потреби об'єднує відповіді від кількох сервісів для клієнтів.
- **Обмеження навантаження:** Реалізує механізми rate-limiting та захист від DDoS.

#### Взаємодія:

- Отримує запити від клієнтів і спрямовує їх до відповідних сервісів через REST/GraphQL API.
- Забезпечує ізоляцію зовнішнього інтерфейсу від внутрішньої логіки системи.

## 2 service-user (User & Auth Service)

- Управління обліковими записами користувачів, їхньою автентифікацією та сесіями.

### Основні особливості:

- **Реєстрація та вхід:** Створення облікових записів, хешування паролів, видача токенів (JWT, OAuth2).
- **Управління профілем:** Зберігання даних користувача (email, профіль, wallet\_address, налаштування) у базі (наприклад, PostgreSQL або MongoDB).
- **Сесійний менеджмент:** Контроль активних сесій, оновлення токенів, валідація прав доступу.

### Взаємодія:

- Отримує запити від API Gateway.
- Співпрацює із service-transaction для оновлення фінансової інформації (наприклад, балансу) та з іншими сервісами, які потребують даних про користувача через стандартизовані API.

## 3 service-core (Service-Core)

- Централізоване управління всіма операціями з картками, аукціонами та ставками із єдиним доступом до основної бази даних.

### Основні особливості:

- **CRUD для карток:** Створення, редагування, видалення та перегляд карток. Зберігає деталі картки: назву, рідкість, зображення, власника, статус (available, in\_auction, sold).

- **Аукціони та ставки:** Організація аукціонів, прийом ставок, обробка логіки торгів (перевірка мінімальних підвищень, валідація ставок, визначення переможця).
- **Централізований доступ:** Єдиний API для операцій з даними, що ізолює базу даних від прямого доступу інших сервісів.
- **Публікація подій:** Відправляє події (наприклад, `auction_created`, `new_bid`, `auction_closed`) через Event Bus для асинхронної взаємодії.

#### **Взаємодія:**

- Приймає запити від API Gateway для операцій з картками та аукціонами.
- Інші сервіси (Transaction & Smart Contract Integration, Notification) отримують необхідні дані та сповіщення через API або Event Bus.
- Забезпечує єдиний канал доступу до бази даних для всіх операцій з бізнес-логікою, що стосується колекційних карток.

#### **4 service-transaction (Transaction & Smart Contract Integration Service)**

- Обробка фінансових операцій і інтеграція зі смарт-контрактами для автоматизації критичних транзакцій.

#### **Основні особливості:**

- **Фінансові операції:** Обробка переказів, платежів за картки, купівель/продажів, лотерейних транзакцій.
- **Ескроу та токенизація:** Використання смарт-контрактів для блокування коштів до виконання умов угоди та автоматичного переведення власності.
- **Інтеграція з blockchain:** Виклик смарт-контрактів для запису змін власності, валідація транзакцій, взаємодія із зовнішніми blockchain API/SDK.



- **Аудит та звітність:** Зберігання історії транзакцій для подальшого аналізу та аудиту.

#### **Взаємодія:**

- Отримує події з service-core (наприклад, завершення аукціону) через Event Bus.
- Взаємодіє з service-user для оновлення фінансових даних користувачів.
- Викликає зовнішні API блокчейну для проведення операцій згідно з умовами смарт-контрактів.

#### **5 service-notification (Notification Service)**

- Централізовано інформувати користувачів про важливі події та зміни в системі.

#### **Основні особливості:**

- **Підписка на події:** Отримує повідомлення від Event Bus (наприклад, auction\_closed, new\_bid, transaction\_completed).
- **Надсилання повідомлень:** Відправляє email, push-повідомлення, SMS тощо, використовуючи стандартні канали зв'язку.
- **Гнучкість:** Налаштовувані шаблони повідомлень, логіка повторної доставки, можливість інтеграції з сторонніми сервісами для розсилки.

#### **Взаємодія:**

- Підписується на події, що публікуються service-core та service-transaction.
- Відправляє повідомлення користувачам через API Gateway або через власний інтерфейс.

- Забезпечує оперативне інформування користувачів про зміни статусу аукціонів, ставок та фінансових операцій.
- 

## Інфраструктурні Компоненти

- **Event Bus / Message Broker:**

Забезпечує асинхронну взаємодію між сервісами через публікацію та підписку на події. Допомогає розвантажити сервіси та підвищити стійкість системи (наприклад, RabbitMQ, Apache Kafka).

- **Service Discovery & Configuration Management:**

Автоматичне виявлення сервісів у кластері (наприклад, Kubernetes, Consul) та централізоване управління конфігураціями для забезпечення узгодженості серед всіх компонентів.

---

## Загальний потік взаємодії

1. **Запит від клієнта:**

Клієнт надсилає запит через **service-gateway**, який перевіряє автентифікацію та маршрутизує запит до потрібного сервісу.

2. **User & Auth Service:**

Обробляє реєстрацію, вхід та управління профілями користувачів, надаючи токени для доступу до системи.

3. **Service-Core:**

Виконує операції з картками, аукціонами та ставками, забезпечуючи централізований доступ до основної бази даних. Публікує події про важливі зміни.

4. **Transaction & Smart Contract Integration Service:**

Отримує події про завершення аукціонів, ініціює фінансові

транзакції та взаємодіє зі смарт-контрактами для забезпечення прозорості операцій.

#### 5. **Notification Service:**

Підписується на події з Event Bus та оперативно інформує користувачів про зміни в системі.

---

Ця архітектура з централізованим доступом до бази даних через **service-core** дозволяє чітко розділити бізнес-логіку на модулі:

- **API Gateway** забезпечує єдиний вхід для зовнішніх запитів із базовою безпекою.
- **User & Auth Service** управляє користувачами та сесіями.
- **Service-Core** концентрує всі операції з картками, аукціонами та ставками, ізольовано працюючи з базою даних.
- **Transaction & Smart Contract Integration Service** відповідає за фінансові операції та інтеграцію зі смарт-контрактами.
- **Notification Service** централізовано інформує користувачів про ключові події.

Інфраструктурні компоненти сприяють асинхронній взаємодії, забезпечують масштабованість та стабільність системи. Такий підхід дозволяє легко адаптувати систему під зростання навантаження та змінювати бізнес-логіку, не порушуючи цілісність архітектури.