САМОСТІЙНА РОБОТА № 2

Розробка децентралізованого застосунку (DApp) на основі смарт-контрактів Solidity та React

1. Мета роботи

Отримати практичні навички розробки децентралізованих застосунків, створення та тестування смарт-контрактів мовою Solidity, використання стандарту ERC-20, підключення криптовалютного гаманця та організації взаємодії клієнтської частини з блокчейном.

Додатковою метою є ознайомлення з основними загрозами безпеці смарт-контрактів та принципами взаємодії з DeFi-протоколами.

2. Програмне забезпечення

Для виконання роботи необхідно використовувати:

- Remix IDE – середовище розробки Solidity;
- Solidity – мова програмування смарт-контрактів;
- MetaMask – криптовалютний гаманець;
- Ganache або Hardhat Network – локальна блокчейн-мережа;
- Node.js;
- React;
- Ethers.js;
- Visual Studio Code.

Для розробки інтерфейсу дозволяється використовувати HTML, CSS та JavaScript.

3. Теоретичні відомості

Смарт-контракт – це програмний код, який розгортається в блокчейні та автоматично виконує визначені правила взаємодії між учасниками.

Мова Solidity використовується для створення смарт-контрактів в екосистемі Ethereum.

Стандарт ERC-20 визначає базовий інтерфейс взаємозамінних токенів. Основними функціями стандарту є:

- `totalSupply()` – загальна кількість токенів;
- `balanceOf()` – баланс користувача;
- `transfer()` – передача токенів;
- `approve()` – надання дозволу на використання токенів;
- `transferFrom()` – передача токенів від імені власника;
- `allowance()` – перевірка дозволеного обсягу токенів.

Приклад базового смарт-контракту:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract SimpleStorage {
    uint256 private storedValue;

    function setValue(uint256 value) public {
        storedValue = value;
    }

    function getValue() public view returns (uint256) {
        return storedValue;
    }
}
```

Для взаємодії React-застосунку зі смарт-контрактом використовується бібліотека Ethers.js.

Приклад підключення гаманця:

```
import { BrowserProvider } from "ethers";

async function connectWallet() {
    if (!window.ethereum) {
        throw new Error("MetaMask не встановлено");
    }

    const provider = new BrowserProvider(
        window.ethereum
    );

    const signer = await provider.getSigner();

    const address = await signer.getAddress();

    console.log("Адреса:", address);

    return { provider, signer, address };
}
```

4. Загальне завдання

Розробити функціональний децентралізований застосунок відповідно до індивідуального варіанта.

Застосунок повинен складатися з двох основних частин:

1. Смарт-контракту, який реалізує бізнес-логіку.
2. Клієнтського вебінтерфейсу, який забезпечує взаємодію користувача зі смарт-контрактом.

Завдання 1. Розробка смарт-контракту.

Необхідно:

- створити контракт мовою Solidity;
- реалізувати функції відповідно до обраного варіанта;
- передбачити перевірку коректності вхідних даних;
- використовувати модифікатори доступу;
- реалізувати події `event` для важливих операцій;
- передбачити обробку помилок через `require`, `revert` або `custom errors`.

Завдання 2. Реалізація токена ERC-20.

Створити окремий токен для використання в застосунку.

Токен повинен підтримувати:

- назву;
- символ;
- загальну емісію;
- перегляд балансу;
- передачу токенів;
- перевірку дозволів на використання токенів.

Дозволяється використовувати OpenZeppelin Contracts для реалізації стандарту ERC-20. Основна бізнес-логіка індивідуального смарт-контракту повинна бути написана самостійно.

Завдання 3. Розгортання смарт-контрактів.

Необхідно:

- скомпілювати контракти в Remix IDE;
- усунути помилки компіляції;
- розгорнути контракти в локальній мережі або тестовому середовищі;
- отримати адресу розгорнутого контракту;
- зберегти ABI для використання у React.

Завдання 4. Розробка клієнтської частини.

Створити React-застосунок, який повинен забезпечувати:

- підключення MetaMask;

- відображення адреси користувача;
- відображення поточного балансу токенів;
- виконання основних операцій відповідно до варіанта;
- відображення результатів транзакцій;
- обробку помилок взаємодії зі смарт-контрактом.

Завдання 5. Перевірка безпеки.

Перевірити програмну реалізацію на наявність типових вразливостей:

- Reentrancy;
- некоректне керування доступом;
- помилки перевірки вхідних даних;
- арифметичні помилки;
- некоректну роботу з дозволами токенів;
- повторне виконання операцій.

Для кожної виявленої проблеми необхідно описати причину її виникнення та спосіб усунення.

5. Індивідуальні варіанти завдань

Варіант 1. Децентралізована система краудфандингу

Розробити DApp для збору коштів на реалізацію проєктів.

Смарт-контракт повинен забезпечувати:

- створення кампаній збору коштів;
- зазначення назви, опису та цільової суми;
- встановлення кінцевої дати кампанії;
- внесення коштів користувачами;
- перегляд поточного обсягу зібраних коштів;
- повернення внесків у разі недосягнення мети;
- отримання коштів автором після успішного завершення кампанії.

React-інтерфейс повинен містити перелік кампаній, інформацію про зібрані кошти та кнопки внесення коштів.

Додатково: реалізувати токенізовану систему внесків, у якій внески здійснюються через ERC-20 токен.

Варіант 2. Децентралізована система електронного голосування

Розробити DApp для проведення голосувань із використанням смарт-контракту.

Смарт-контракт повинен забезпечувати:

- створення голосування;
- додавання кандидатів;
- реєстрацію учасників;
- перевірку права голосу;
- можливість голосування лише один раз;
- автоматичний підрахунок результатів;
- завершення голосування.

React-інтерфейс повинен забезпечувати вибір кандидата, підтвердження транзакції та перегляд результатів.

Додатково: реалізувати обмеження доступу до створення голосувань для адміністратора.

Варіант 3. Децентралізований маркетплейс цифрових товарів

Розробити DApp для продажу цифрових товарів із використанням блокчейну.

Смарт-контракт повинен забезпечувати:

- реєстрацію товарів;
- додавання назви, опису та вартості;
- реєстрацію продавця;
- купівлю товарів;
- перевірку достатності коштів;

- збереження історії продажів;
- підтвердження власності покупця.

React-інтерфейс повинен містити каталог товарів, картки товарів, форму купівлі та історію придбань.

Додатково: реалізувати комісію платформи з кожної успішної операції.

Варіант 4. Децентралізована система управління нерухомістю

Розробити DApp для реєстрації об'єктів нерухомості та операцій оренди.

Смарт-контракт повинен забезпечувати:

- реєстрацію об'єктів нерухомості;
- зазначення власника, адреси та вартості оренди;
- публікацію об'єктів для оренди;
- створення договорів оренди;
- оплату оренди через токени;
- перевірку терміну дії договору;
- завершення договору.

React-інтерфейс повинен містити перелік доступних об'єктів, інформацію про власників та інструменти оформлення оренди.

Додатково: реалізувати заставний депозит, який повертається орендарю після завершення договору за умови виконання його умов.

Варіант 5. Децентралізована система сертифікації документів

Розробити DApp для перевірки справжності цифрових документів.

Смарт-контракт повинен забезпечувати:

- реєстрацію документів;
- збереження криптографічного хешу документа;
- зазначення автора або організації-видавця;
- збереження дати видачі;
- перевірку документа за хешем;
- відкликання документа уповноваженим користувачем;
- перевірку статусу документа.

React-інтерфейс повинен дозволяти реєструвати документ, вводити його хеш і перевіряти справжність.

Додатково: реалізувати перевірку документів із використанням SHA-256 на клієнтській стороні.

Варіант 6. Децентралізована система обліку благодійних внесків

Розробити DApp для збору та контролю благодійних коштів.

Смарт-контракт повинен забезпечувати:

- створення благодійних проєктів;
- встановлення цільової суми;
- приймання внесків через ERC-20;
- облік внесків кожного користувача;
- перегляд загального обсягу зібраних коштів;
- виведення коштів уповноваженим отримувачем;
- збереження історії операцій.

React-інтерфейс повинен відображати благодійні кампанії, прогрес збору коштів та історію внесків.

Додатково: реалізувати механізм обмеження використання коштів лише авторизованим отримувачем.

Варіант 7. Децентралізована система стейкінгу токенів

Розробити DApp для блокування токенів користувачами з метою отримання винагороди.

Смарт-контракт повинен забезпечувати:

- внесення токенів у стейкінг;

- облік частки кожного користувача;
- обчислення винагороди;
- отримання винагороди;
- виведення заблокованих токенів;
- встановлення мінімального періоду блокування;
- перевірку доступного для виведення балансу.

React-інтерфейс повинен відображати кількість застейканих токенів, доступну винагороду та історію операцій.

Додатково: реалізувати штрафування за порушення встановлених правил стейкінгу.

Варіант 8. Спрощений децентралізований обмінник токенів

Розробити DApp для обміну двох ERC-20 токенів через смарт-контракт.

Смарт-контракт повинен забезпечувати:

- створення пулу ліквідності;
- внесення двох типів токенів;
- обчислення обмінного курсу;
- здійснення операції обміну;
- розрахунок комісії;
- перевірку достатності ліквідності;
- виведення частки ліквідності.

React-інтерфейс повинен забезпечувати вибір токенів, введення суми, попередній розрахунок обміну та підтвердження транзакції.

Додатково: реалізувати розрахунок ціни за спрощеною моделлю постійного добутку `x * y = k`.

Варіант 9. Децентралізована система кредитування

Розробити DApp для моделювання кредитування з використанням токенів.

Смарт-контракт повинен забезпечувати:

- внесення коштів у кредитний пул;
- створення заявок на кредит;
- встановлення суми та відсоткової ставки;
- перевірку застави;
- видачу кредиту;
- повернення кредиту з відсотками;
- перевірку термінів погашення.

React-інтерфейс повинен відображати доступну ліквідність, заявки, видані кредити та стан погашення.

Додатково: реалізувати ліквідацію застави у випадку порушення встановлених умов кредитування.

Варіант 10. Децентралізована система управління цифровими активами

Розробити DApp для обліку цифрових активів користувачів.

Смарт-контракт повинен забезпечувати:

- створення цифрових активів;
- реєстрацію власника;
- передачу активів між користувачами;
- збереження ідентифікатора та опису активу;
- перевірку права власності;
- збереження історії передачі;
- перегляд активів конкретного користувача.

React-інтерфейс повинен відображати перелік активів, поточних власників та історію передач.

Додатково: реалізувати механізм блокування активу адміністратором у разі виявлення підозрілої операції.

6. Вимоги до програмної реалізації

Розроблений DApp повинен містити:

1. Смарт-контракти:

- основний контракт із бізнес-логікою відповідно до варіанта;
- ERC-20 токен;
- модифікатори доступу;
- події для основних операцій;
- перевірки коректності даних.

2. Клієнтську частину:

- React-застосунок;
- підключення MetaMask;
- використання Ethers.js;
- відображення адреси користувача;
- відображення балансу токенів;
- взаємодію з функціями смарт-контракту;
- відображення результатів транзакцій.

3. Тестування та безпеку:

- перевірку успішних операцій;
- перевірку відхилених операцій;
- тестування прав доступу;
- перевірку типових вразливостей;
- виправлення виявлених проблем.

Для перевірки безпеки дозволяється використовувати Slither та інші інструменти статичного аналізу. Їх використання не замінює самостійного аналізу коду.

7. Контрольні експерименти

Під час виконання роботи необхідно провести такі експерименти:

№	Експеримент	Очікуваний результат
1	Компіляція смарт-контракту	Контракт успішно скомпільовано
2	Розгортання контракту	Отримано адресу контракту
3	Підключення MetaMask	Визначено адресу користувача
4	Отримання балансу ERC-20	Відображено правильний баланс
5	Виконання основної операції	Транзакцію успішно виконано
6	Операція з недостатнім балансом	Транзакцію відхилено
7	Операція без відповідних прав	Доступ заборонено
8	Перевірка безпеки	Виявлені проблеми усунено або обґрунтовано