Лабораторна робота №4: Розробка децентралізованих застосунків (DApps)

1. Теоретична частина

1.1. Архітектура DApp

У традиційному вебі (Web2) ми використовуємо клієнт-серверну архітектуру: Frontend звертається до Backend-сервера (API), який взаємодіє з базою даних.

У **DApp (Web3)** концепція змінюється: **Блокчейн — це ваш Backend**.

- **Frontend:** React/Vue/Vanilla JS додаток, що працює в браузері користувача.
- **Blockchain:** Смартконтракти, що зберігають стан (state) та логіку.
- **Provider:** "Вузол" (Node), через який фронтенд спілкується з мережею.

Порівняння:

Характеристика	Традиційний застосунок (Web2)	DApp (Web3)
Зберігання даних	Централізована БД (PostgreSQL, MongoDB)	Децентралізований реєстр (Ethereum)
Автентифікація	Login/Password, OAuth (Google/FB)	Публічний ключ (Гаманець)
Логіка	Серверний код (Node.js, Python)	Смартконтракти (Solidity)
Хостинг	AWS, Azure, Google Cloud	IPFS, Arweave (децентралізація)

1.2. Web3-бібліотеки: Ethers.js та Web3.js

Щоб JavaScript зміг "прочитати" блокчейн, йому потрібен перекладач. Цю роль виконують бібліотеки **Ethers.js** (сучасніша, легша) та **Web3.js** (класична).

Ключові поняття:

1. **Provider:** Об'єкт, що дозволяє тільки **читати** дані з блокчейну (баланс, стан змінних). Він не потребує приватного ключа.
2. **Signer:** Об'єкт, що має доступ до приватного ключа (зазвичай через гаманець). Він дозволяє **підписувати транзакції** та змінювати стан блокчейну.

1.3. Інтеграція з гаманцями (Metamask)

Metamask діє як міст. Він вприскує (inject) об'єкт window.ethereum у ваш браузер. Цей об'єкт дозволяє сайту просити у користувача дозвіл на підключення гаманця та підпис транзакцій.

1.4. Середовища розробки: Hardhat vs Foundry

Хоча **Remix IDE** чудовий для швидких тестів, професійна розробка ведеться локально через:

- **Hardhat:** Екосистема на базі JS/TS. Найбільша кількість плагінів, зручна інтеграція з фронтендом.
- **Foundry:** Написаний на Rust. Надзвичайно швидкий. Тести пишуться на **Solidity**, що дозволяє розробнику не перемикатися на іншу мову.

Чому не Remix? Локальні середовища дозволяють:

- Використовувати Git для контролю версій.
- Писати складні автоматизовані тести.
- Використовувати змінні оточення (.env) для безпеки.

1.5. Мережі та Testnets

Розгортання в Mainnet коштує реальних грошей. Для навчання ми використовуємо:

- **Testnets (Sepolia, Holesky):** Публічні мережі, що копіюють поведінку Ethereum, але використовують безкоштовні токени.
- **Faucets (Крани):** Сервіси, які видають невелику кількість тестового ETH на вашу адресу для оплати газу.

2. Практична частина

Завдання 1: Налаштування Hardhat

Створіть папку проєкту та ініціалізуйте її:

mkdir my-dapp && cd my-dapp

npm init -y

npm install --save-dev hardhat

npx hardhat # Оберіть "Create a JavaScript project"

Структура папок:

- /contracts: Ваші Solidity файли.

- /scripts: Скрипти для деплою.
- /test: Файли тестів.
- hardhat.config.js: Головний файл конфігурації.

Завдання 2: Тестування (JavaScript + Chai)

Тести — це єдиний спосіб переконатися в безпеці коду до деплою, адже смартконтракт неможливо змінити після публікації.

```
const { expect } = require("chai");


describe("SimpleStorage", function () {
  it("Should return the new value once it's changed", async function () {
    const Storage = await ethers.getContractFactory("SimpleStorage");
    const storage = await Storage.deploy();
    await storage.deployed();


    const updateTx = await storage.set(42);
    await updateTx.wait();


    expect(await storage.get()).to.equal(42);
  });
});
```

Запуск: npx hardhat test

Завдання 3: Фронтенд-інтеграція (Ethers.js)

Приклад простого HTML/JS коду для взаємодії з гаманцем:

```
<!DOCTYPE html>
<html>
<body>
  <button id="connectBtn">Connect Wallet</button>
  <p id="accountArea"></p>
```

```
  <script src="[https://cdn.ethers.io/lib/ethers-5.2.umd.min.js](https://cdn.ethers.io/lib/ethers-5.2.umd.min.js)"></script>
  <script>
    const connectBtn = document.getElementById('connectBtn');

    connectBtn.onclick = async () => {
      if (window.ethereum) {
        // 1. Запит на підключення
        const accounts = await window.ethereum.request({ method: 'eth_requestAccounts' });
        document.getElementById('accountArea').innerText = "Connected: " + accounts[0];

        // 2. Ініціалізація Ethers
        const provider = new ethers.providers.Web3Provider(window.ethereum);
        const signer = provider.getSigner();

        // 3. Читання балансу
        const balance = await provider.getBalance(accounts[0]);
        console.log("Balance in ETH:", ethers.utils.formatEther(balance));
      } else {
        alert("Please install Metamask!");
      }
    };
  </script>
</body>
</html>
```

Завдання 4: Деплой у тестову мережу

Для деплою в Sepolia оновіть hardhat.config.js:

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

module.exports = {
  solidity: "0.8.19",
  networks: {
    sepolia: {
      url: process.env.SEPOLIA_RPC_URL,
      accounts: [process.env.PRIVATE_KEY]
    }
  }
};
```

Запуск: npx hardhat run scripts/deploy.js --network sepolia

3. Контрольні запитання

1. Яка принципова різниця між Hardhat Network (local) та публічною тестовою мережею (наприклад, Sepolia)?
2. Навіщо потрібен файл .env при розробці DApp і чому його ніколи не можна завантажувати в публічний репозиторій?
3. Як DApp дізнається про те, що користувач змінив акаунт або мережу в Metamask (підказка: події window.ethereum)?
4. У чому різниця між викликом функції смартконтракту через call та надсиланням transaction?
5. Що таке ABI (Application Binary Interface) і чому фронтенду необхідно знати його для взаємодії з контрактом?