Multichain dApps

Цель проекта - познакомиться с технологиями создания распределенных приложений, которые могут взаимодействовать с несколькими блокчейнами, в том числе через протоколы второго уровня. В основе проекта лежат технологии (next.js)[https://nextjs.org/] и (viem)[https://viem.sh/].

1 Подготовка next.js приложения

Создайте новый проект `next.js` с настройками по-умолчанию:

```
npx create-next-app@latest dapp
```

Выбирайте ответ `yes` для этих пунктов:

```
Typescript
ESLint
Tailwind
App Router (preferably)
```

Установите зависимости приложения и `viem`:

```
cd app
npm i
npm i viem
```

Проверьте работу приложения:

```
npm run dev
```

Подключите стили в `app/globals.css`

```
.card{
  margin: 16px;
  background-color: #fff;
  border: 1px solid #cbd5e0;
  box-shadow: 0 1px 2px 0 rgba(0, 0, 0, 0.05);
  border-radius: 0.375rem;
  padding: 32px;
  min-width: 600px;
}

input {
  padding: 10px 0;
  border-radius: 0.125rem;
  border: 1px solid rgb(226, 232, 240);
  background-color: #fdfdfe;
  padding-inline-start: 0.75rem;
  font-size: 0.875rem;
  margin: 0 0 16px 8px;
  min-width: 400px;
}
```

2 Подключение библиотеки viem

В папке `app` добавьте файл `client.ts`, инкапсулирующий логику создания клиента.

```
import { createWalletClient, createPublicClient, custom, http } from "v
import { sepolia } from "viem/chains";
import "viem/window";

export function ConnectPublicClient() {
    let transport;
    if (window.ethereum) {
        transport = custom(window.ethereum);
    } else {
        transport = http();
    }

    const publicClient = createPublicClient({
        chain: sepolia,
        transport: transport,
    });

    return publicClient;
}

export function ConnectWalletClient() {
    let transport;
    if (window.ethereum) {
        // EIP-1193 Ethereum Provider JavaScript API
        transport = custom(window.ethereum);
        // or transport = http("https://eth.web3gate.ru:32443/e6c02775.
    } else {
        const errorMessage ="Web3 wallet is not installed. Please insta
        throw new Error(errorMessage);
    }

    const walletClient = createWalletClient({
        chain: sepolia,
        transport: transport,
    });

    return walletClient;
}
```

Перед созданием клиента нужно проверить, что браузер имеет объект `window.ethereum`. В противном случае необработанная ошибка не даст запустить приложение.

3 Подключение web3 кошелька

Создайте компонент для проверки баланса подключенного в браузере кошелька. В папке `app` наполните содержимое файла `walletComponent.tsx`:

```
"use client";
import { useState } from "react";
import { ConnectWalletClient, ConnectPublicClient } from "./client";

export default function WalletComponent() {
  const [address, setAddress] = useState("");
 const [balance, setBalance] = useState(BigInt(0));

  async function handleClick() {
    try {
      const walletClient = ConnectWalletClient();
   const publicClient = ConnectPublicClient();

      const [address] = await walletClient.getAddresses();
   const balance: bigint = await publicClient.getBalance({ address });

      setAddress(address);
   setBalance(balance);
    } catch (error) {
      alert(`Transaction failed: ${error}`);
    }
  }

  return (
    <div className="card">
      <Status address={address} balance={balance} />
      <button className="px-8 py-2 rounded-md flex flex-row items-cente
        onClick={handleClick} >
        <h1 className="mx-auto">Connect Wallet</h1>
      </button>
    </div>
  );
}

function Status({
  address,
  balance,
}: {
  address: string | null;
  balance: BigInt;
}) {
  if (!address) {
    return (
      <div className="flex items-center">
        <div className="border bg-red-600 border-red-600 rounded-full w
        <div>Disconnected</div>
      </div>
    );
  }

  return (
    <div className="flex items-center w-full">
      <div className="border bg-green-500 border-green-500 rounded-full
```

```
      <div className="text-s md:text-s">
        {address} <br /> <b>Balance:</b> {balance.toString()} <b>Wei</b
      </div>
    </div>
  );
}
```

Измените разметку основной страницы `app/page.tsx`:

```
import TokenComponent from "./tokenComponent";
import TransactionComponent from "./transactionComponent";
import WalletComponent from "./walletComponent";

export default function Home() {
      return (
            <main className="min-h-screen">
                  <div className="flex flex-col items-center justify-ce
                              <WalletComponent />
                </div>
            </main>
      );
}
```

Запустите приложение и протестируйте подключение кошелька.

4 Совершение транзакции

Получите криптовалюту в тестовой сети `Sepolia` через специальный сервис или с другого аккаунта.

Реализуйте компонент для совершения транзакций средствами библиотеки `viem`.

Создайте файл `app/transactionComponent.tsx` со следующим содержимым:

```
"use client";
import { useState } from "react";
import { parseGwei } from "viem";
import { ConnectWalletClient } from "./client";

export default function TransactionComponent() {
   const [amount, setAmount] = useState("");
   const [recipient, setRecipient] = useState("");

   const setValue = (setter:any) => (evt:any) => setter(evt.target.valu

   async function handleClick() {
      try {
         const walletClient = ConnectWalletClient();
         const [address] = await walletClient.getAddresses();
         const hash = await walletClient.sendTransaction({
            account: address,
            to: recipient,
            value: parseGwei(amount),  // GWei
```

```
        });
        alert(`Transaction successful. Transaction Hash: ${hash}`);
      } catch (error) {
          alert(`Transaction failed: ${error}`);
      }
   }

   return (
   <div className="card">
      <label>
      Amount:
      <input
         placeholder="GWei"
         value={amount}
         onChange={setValue(setAmount)}
      ></input>
      </label>
      <br />

      <label>
      Recipient:
      <input
         placeholder="Address"
         value={recipient}
         onChange={setValue(setRecipient)}
      ></input>
   </label>
      <button
         className="px-8 py-2 rounded-md flex flex-row items-center jus
         onClick={handleClick}>
         Send Transaction
      </button>
   </div>

   );
}
```

Совершите тестовую транзакцию, передав несколько GWei.

5 Взаимодействие со смарт-контрактом

Изучите размещенные смарт-контракты в сети Ethereum, например (Sepolia)[https://sepolia.etherscan.io/contractsVerified]

Мы рассмотрим взаимодействие со смарт-контрактом ERC721, размещенным по (адресу)[https://sepolia.etherscan.io/address/0xae2a37b60b7af7fcca8167df617f82a34f22719c]

Взаимодействие будет заключаться в исполнении функций `symbol()` и `name()` для чтения информации о токене. Также будет показано как вызываются функции с аргументами, например `owner_of(token_id)`. Эта функция возвращает адрес владельца токена (NFT) с указанным id.

Создайте файл `abi.ts`, в который вставьте содержимое бинарного интерфейса смарт-контракта:

```
export const contractAbi = [...contract abi...] as const;
```

Создайте компонент `app/tokenComponent.tsx` со следующим содержимым

```
"use client";
import { getContract, Address } from "viem";
import { contractAbi } from "./abi";
import { ConnectWalletClient } from "./client";
import { useState } from "react";

export default function TokenComponent() {
  const [contractAddress, setContractAddress] = useState("");
  const [tokenId, setTokenId] = useState();

  const setValue = (setter:any) => (evt:any) => setter(evt.target.value

  const walletClient = ConnectWalletClient();

  async function buttonClick() {
    const checkedAddress = contractAddress as Address;

    const contract = getContract({
      address: checkedAddress,
      abi: contractAbi,
      client: walletClient,
    });

    console.log("Connected to Contract: ", contract);

    const symbol = await contract.read.symbol();
    const name = await contract.read.name();

    console.log(`Symbol: ${symbol}\nName: ${name}\n`);

    const token_id = BigInt(tokenId);
    const owner = await contract.read.ownerOf([token_id]);

    alert(`Symbol: ${symbol}\nName: ${name}\nOwner of token_id = ${toke

  }

    return (
      <div className="card">
        <label>
        Address:
        <input
          placeholder="Smart Contract Instance"
          value={contractAddress}
          onChange={setValue(setContractAddress)}
        ></input>
```

```
        </label>
        <br />

        <label>Token Id:
        <input placeholder="1" value={tokenId} onChange={setValue(setTo
        </label>
          <button
            className="px-8 py-2 rounded-md flex flex-row items-center
            onClick={ buttonClick }>
            <h1 className="text-center">Token Info</h1>
          </button>
      </div>
    );
}
```

Добавьте компонент взаимодействия с токеном на главную страницу `page.tsx`

Протестируйте взаимодействие со смарт-контрактом.

Задание на самостоятельную работу:

Найдите другой размещенный смарт-контракт и напишите компонент для взаимодействия с его abi.

Шаблон для распределенных приложений

(Scaffold-eth-2)[https://github.com/scaffold-eth/scaffold-eth-2] это шаблонный проект для построения собственных распределенных приложений. Он базируется на изученных нами технологиях. Изучите его и используйте для построения своего выпускного проекта!