Integração do Foundry na Aplicação de Tokenização

Este documento descreve o processo de integração do Foundry, uma toolchain de desenvolvimento de smart contracts, na nossa aplicação de tokenização de ativos reais.

Índice

Visão Geral

O Foundry é uma toolchain de desenvolvimento de smart contracts escrita em Rust, que oferece um conjunto de ferramentas para compilar, testar, implantar e interagir com contratos inteligentes. A integração do Foundry na nossa aplicação de tokenização permite:

- Desenvolvimento rápido e eficiente de contratos inteligentes
- Testes robustos e abrangentes
- Implantação simplificada em diferentes redes
- Melhor integração com a aplicação web existente

Instalação e Configuração

Pré-requisitos

- Node.js e npm (já instalados no projeto)
- Git (já instalado)

Instalação do Foundry

```
# Instalar o Foundry
curl -L https://foundry.paradigm.xyz | bash
source ~/.bashrc
foundryup

# Verificar a instalação
forge --version
anvil --version
cast --version
```

Configuração do Projeto

```
# Inicializar o projeto Foundry
mkdir onchain
cd onchain
forge init . --no-git
```

```
# Instalar dependências
forge install OpenZeppelin/openzeppelin-contracts@v4.9.6
```

Estrutura do Projeto

A estrutura do projeto após a integração do Foundry é a seguinte:

```
tokenizacao-app/
├── app/                      # Aplicação Next.js existente
│   ├── lib/
│   │   └── contracts/        # Integração com contratos (ABIs, hooks, etc.)
│   └── ...
├── onchain/                  # Diretório para desenvolvimento de contratos
│   ├── lib/                  # Dependências (OpenZeppelin, etc.)
│   ├── script/               # Scripts de implantação
│   ├── src/                  # Contratos inteligentes
│   ├── test/                 # Testes dos contratos
│   └── foundry.toml          # Configuração do Foundry
└── foundry-integration.md    # Esta documentação
```

Contratos Inteligentes

Foram desenvolvidos três contratos principais para a aplicação:

AssetToken (ERC-721)

Contrato para representar ativos reais tokenizados como NFTs.

Funcionalidades principais: - Criação de tokens representando ativos reais - Armazenamento de metadados e informações do ativo - Verificação de ativos por autoridades - Gerenciamento de propriedade dos tokens

Marketplace

Contrato para gerenciar o marketplace de ativos tokenizados.

Funcionalidades principais: - Listagem de ativos para venda - Compra de ativos listados - Cancelamento de listagens - Cobrança de taxas de transação - Gerenciamento de pagamentos

Waitlist

Contrato para gerenciar a lista de espera para tokenização de ativos.

Funcionalidades principais: - Adição de usuários à lista de espera - Aprovação de usuários por administradores - Remoção de usuários da lista - Consulta paginada da lista de espera

Testes

Foram desenvolvidos testes abrangentes para cada contrato, garantindo que todas as funcionalidades funcionem conforme esperado.

Executando os Testes

```
cd onchain
forge test
```

Os testes incluem: - Testes de unidade para cada função - Testes de integração entre

contratos - Testes de casos de erro (revert)

Implantação

Scripts de Implantação

Foram criados scripts para facilitar a implantação dos contratos em diferentes ambientes:

- `DeployContracts.s.sol`: Script Solidity para implantação dos contratos
- `start_anvil.sh`: Script para iniciar um nó local Anvil para desenvolvimento
- `deploy_local.sh`: Script para implantar os contratos no Anvil local

Implantação Local

```
# Iniciar o Anvil
./onchain/script/start_anvil.sh

# Em outro terminal, implantar os contratos
./onchain/script/deploy_local.sh
```

Implantação em Testnet/Mainnet

Para implantar em redes de teste ou produção, é necessário configurar as variáveis de ambiente:

```
# Configurar chave privada (não armazenar em repositórios!)
export PRIVATE_KEY=sua_chave_privada_aqui

# Implantar na Sepolia (testnet)
forge script script/DeployContracts.s.sol:DeployContracts --rpc-url
        https://sepolia.infura.io/v3/YOUR_INFURA_KEY --broadcast --verify

# Implantar na Mainnet
forge script script/DeployContracts.s.sol:DeployContracts --rpc-url
        https://mainnet.infura.io/v3/YOUR_INFURA_KEY --broadcast --verify
```

Integração com Next.js

A integração com a aplicação Next.js existente foi feita através de hooks React que facilitam a interação com os contratos.

Estrutura de Arquivos

- `app/lib/contracts/config.ts`: Configuração dos endereços dos contratos e RPC URLs
- `app/lib/contracts/hooks.ts`: Hooks React para interagir com os contratos
- `app/lib/contracts/index.ts`: Exportações para facilitar a importação
- `app/lib/contracts/*.abi.json`: ABIs dos contratos

Hooks Disponíveis

- `useEthers()`: Gerencia a conexão com a carteira e o provedor Ethereum
- `useAssetToken()`: Interação com o contrato AssetToken
- `useMarketplace()`: Interação com o contrato Marketplace
- `useWaitlist()`: Interação com o contrato Waitlist

Exemplo de Uso

```
import { useAssetToken, useMarketplace, useWaitlist } from '../lib/contracts';
```

```
function MyComponent() {
  const { getAsset, mintAsset } = useAssetToken();
  const { listAsset, buyAsset } = useMarketplace();
  const { joinWaitlist, isApproved } = useWaitlist();

  // Usar as funções conforme necessário
  // ...
}
```

Desenvolvimento Local

Para desenvolvimento local, siga estes passos:

1. Inicie o nó Anvil:

   ```
   ./onchain/script/start_anvil.sh
   ```

2. Implante os contratos:

   ```
   ./onchain/script/deploy_local.sh
   ```

3. Inicie a aplicação Next.js:

   ```
   cd app
   npm run dev
   ```

4. Conecte o MetaMask ao Anvil:
   - RPC URL: http://localhost:8545
   - Chain ID: 31337
   - Moeda: ETH

5. Importe uma conta de teste:
   - Chave privada: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80 (primeira conta do Anvil)

Próximos Passos

- Implementar verificação de contratos no Etherscan
- Adicionar suporte a mais redes (Polygon, Arbitrum, etc.)
- Melhorar a UI para interação com os contratos
- Implementar sistema de notificações para eventos dos contratos
- Adicionar funcionalidades de governança para o marketplace