TP 8: Tokens (150 minutes)

Token Mechanics Across Blockchains - Solana and Ethereum

Duration: 2-3 hours
Objective:

1. Understand and implement token systems on two major blockchains: Solana and Ethereum.
2. Compare the differences in approaches, tooling, and implementation between the two ecosystems.
3. Gain practical experience in token creation, interaction, and use-case design through a real-world scenario.

Decentralized Rewards System - "ChainRewards"

Imagine you're tasked with developing a decentralized rewards platform called **ChainRewards**. The system will reward users with tokens for completing specific tasks, such as making purchases or engaging in eco-friendly activities.

- Users earn **REWARD tokens** for their actions.
- These tokens can be transferred between users or redeemed for rewards.
- The goal is to ensure the system is secure, transparent, and abuse-resistant.

You will implement **ChainRewards** on:

1. **Solana** using SPL tokens and Programs.
2. **Ethereum** using ERC-20 tokens and Smart Contracts.

Part 1: Solana Token Mechanics (80 minutes)

Sub-Part 1.1: Understanding the Use Case and Problem Statement (15 mins)

Guidance:

1. **Breakdown the Use Case**:
   - Why do we need tokens?
   - How do tokens enable rewards and transfers in ChainRewards?
2. **Define Key Requirements**:
   - **Minting**: Tokens are issued based on user actions.
   - **Transfer**: Tokens must be transferable between accounts.
   - **Constraints**:
     - Limit the maximum tokens per user.
     - Prevent unauthorized minting or double transfers.

Challenge:

- Write down the potential **security risks** and **abuse cases**.
- Suggest solutions for ensuring fairness and transparency.

Resource:

- Solana SPL Token Overview.

1.2: Environment Setup and SPL Token Creation (15 mins)

Guidance:

1. **Set Up Environment (should already be done in the previous TP)**:
   - Install Solana CLI, Rust, and Anchor.

    - Set up a local validator or connect to the Devnet.
2. **Create an SPL Token**:
    - Use the Solana CLI to create an account
    - Then, use SPL-TOKEN create and mint a token
    - Transfer tokens between accounts.

Bonus Challenge:

- Write a script using **Solana's JavaScript SDK** to automate token minting.

Resources:

- Solana CLI Docs.
- Spl-token Command-line.

1.3: Writing a Solana Program for Tokens (45 mins)

Guidance:

1. **Program Features**:
    - Automate token minting based on user actions.
    - Allow users to transfer tokens programmatically.

Tasks:

- Write a **Solana Program** in Rust:
    - Define user actions that trigger minting (e.g., completing a task).
    - Validate inputs to prevent abuse.
- Test your program using the Anchor testing framework.

Challenge:

- Add constraints to the program:
    - Require users to reach a **minimum threshold** of actions before tokens are minted.
    - Log all transactions for auditing purposes.

Resources:

- Anchor Framework Docs.
- Solana Program Examples.

Sub-Part 1.4: Testing and Deployment (30 mins)

Guidance:

1. Write Unit Tests:
    - Test minting logic with valid and invalid inputs.
    - Verify transfer functionality.
2. Deploy the Program:
    - Use the Solana CLI to deploy the program to Devnet.
    - Interact with the deployed program using scripts or CLI commands.

Bonus Challenge:

- Extend the program to support **burning tokens** when they are redeemed.

Resources:

- Testing Solana Programs.

Part 2: Ethereum Token Mechanics (80 minutes)

Sub-Part 2.1: Problem Translation to Ethereum (15 mins)

Guidance:

1. **Compare SPL vs ERC-20 Tokens**:

    - **SPL Tokens**: Managed by the token program on Solana.
    - **ERC-20 Tokens**: Fully custom smart contracts on Ethereum.
2. Discuss the differences in mechanics, such as:
    - Gas fees.
    - Ecosystem tooling.

Resource:

- ERC-20 Standard.

2.2: Environment Setup and ERC-20 Token Creation (15 mins)

Guidance:

1. **Set Up Environment**:
    - Install Node.js, Hardhat, and Ganache.
    - Configure MetaMask for a local Ethereum network.
2. **Create ERC-20 Token**:

- Write an ERC-20 contract:

// SPDX-License-Identifier: MIT

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract RewardToken is ERC20 {
// complete the code here
}
```

Task:

- Deploy the contract using Hardhat.

Challenge:

- Add minting logic that only allows specific addresses to mint tokens.

Resources:

- Hardhat Docs.
- OpenZeppelin Contracts.

2.3: Testing and Interaction (30 mins)

Guidance:

1. **Write Unit Tests**:
    - Test minting, transferring, and burning tokens.
    - Verify constraints such as maximum supply.
2. **Deploy and Interact**:
    - Deploy the contract to a local or test Ethereum network.
    - Use Hardhat Console to interact with the contract.

Challenge:

- Extend the contract to include a **redemption function** for burning tokens.

Resources:

- Testing with Hardhat.

2.4: Comparison and Deployment (20 mins)

Guidance:

1. **Discussion**:
    - Compare your experience with Solana and Ethereum.
    - Discuss trade-offs in terms of:
        - Development complexity.
        - Ecosystem tooling.
        - Performance and cost.
2. **Deploy to Public Testnets**:
    - Deploy the ERC-20 contract to Goerli.
    - Share contract addresses with the class for interaction.

Bonus Challenge:

- Build a minimal frontend for interacting with both Solana and Ethereum tokens.

Activité précédente

Chapitre 08 : Les Tokens

Aller à…

Activité suivante

Mini Projet : Monopoly