**ACTIVIDAD PRÁCTICA # 2**

**Preparación del Entorno**

Instalar un entorno de desarrollo local de Ethereum, como [**Ganache CLI**](https://www.npmjs.com/package/ganache-cli) o [**Truffle Develop.**](https://archive.trufflesuite.com/docs/truffle/how-to/debug-test/use-truffle-develop-and-the-console/) **(Siga la guía de instalación)**

Desplegar el contrato inteligente que se va a probar en la red local. Por ejemplo, supongamos que tenemos un contrato simple de token ERC20 llamado MyToken.

// Contrato del token ERC20 básico

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract MyToken {

string public name = "MyToken";

string public symbol = "MTK";

uint256 public totalSupply;

mapping(address => uint256) public balanceOf;

constructor(uint256 \_initialSupply) {

totalSupply = \_initialSupply;

balanceOf[msg.sender] = \_initialSupply;

}

}

**Definición de Escenarios de Prueba**

Identificar los escenarios de prueba que se van a simular, como la transferencia masiva de tokens entre cuentas o la ejecución simultánea de múltiples transacciones de un mismo tipo.

Establecer los parámetros de las pruebas, como el número de transacciones por segundo o la duración total de la prueba.

**Creación de Scripts de Prueba**

Escribir scripts de pruebas automatizadas utilizando Truffle o scripts de JavaScript para simular la ejecución de transacciones en el contrato inteligente.

Por ejemplo, aquí hay un script simple en JavaScript que simula la transferencia masiva de tokens entre cuentas:

const MyToken = artifacts.require("MyToken");

contract("MyToken", async (accounts) => {

it("should transfer tokens between accounts", async () => {

const myTokenInstance = await MyToken.deployed();

const sender = accounts[0];

const receiver = accounts[1];

const amount = web3.utils.toBN(100); // Transfer 100 tokens

// Transfer tokens from sender to receiver

await myTokenInstance.transfer(receiver, amount, { from: sender });

// Verify balances

const senderBalance = await myTokenInstance.balanceOf(sender);

const receiverBalance = await myTokenInstance.balanceOf(receiver);

assert.equal(senderBalance.toString(), "9900", "Sender balance incorrect");

assert.equal(receiverBalance.toString(), "100", "Receiver balance incorrect");

});

});

**Ejecución de Pruebas:**

Ejecutar los scripts de pruebas en el entorno local para simular la carga de trabajo en el contrato inteligente.

Por ejemplo, usando Truffle, ejecutar el comando truffle test para ejecutar las pruebas definidas en los scripts.

**Análisis de Resultados:**

Analizar los resultados de las pruebas para identificar posibles cuellos de botella o problemas de rendimiento en el contrato inteligente.

Comparar las métricas obtenidas con los objetivos de rendimiento establecidos previamente.

**Optimización y Mejoras:**

Realizar ajustes en el contrato inteligente o en la configuración del entorno de pruebas para mejorar el rendimiento y la escalabilidad.

Implementar soluciones para abordar cualquier problema identificado durante el análisis de resultados.

**Documentación y Reporte:**

Documentar los resultados de las pruebas, incluyendo las métricas clave obtenidas y las acciones tomadas para optimizar el rendimiento del contrato.

Preparar un informe detallado que resuma los hallazgos y las recomendaciones para futuras pruebas y mejoras.