**Project Title:** Token-Based Loyalty Program using Smart Contract

**Name:** Aniket Ganesh Singh  
**Roll Number:** 14  
**Course:** M. Tech Computer Engineering  
**Institute:** KJSSE  
**Guide:** Dr. Deepak Sharma

**Abstract**

This project presents a blockchain-based solution to modernize and simplify loyalty programs. Traditional loyalty systems are plagued with issues like lack of transparency, non-transferability, and expiration of reward points. Using Ethereum smart contracts and ERC20 tokens, this project introduces a decentralized and secure way for customers to earn, transfer, and redeem loyalty points.

**Introduction**

Loyalty programs have long been a tool to retain customers and incentivize purchases. However, centralized systems often have limited interoperability, offer non-transferable points, and are susceptible to manipulation or mismanagement. Blockchain technology, with its decentralized and transparent nature, offers a transformative way to handle loyalty programs.

This project develops a basic smart contract on Ethereum that allows users to:

* Earn tokens based on purchases
* Transfer tokens to other users
* Redeem tokens for rewards

**Objective**

* Implement a smart contract for a token-based loyalty program.
* Ensure the transferability and redeemability of tokens.
* Prevent unauthorized or fraudulent transactions.
* Deploy and test using MetaMask and Remix IDE.

**Technology Stack**

* **Solidity:** Programming language for writing smart contracts
* **Remix IDE:** Online development environment for Ethereum
* **MetaMask:** Ethereum wallet to interact with dApps
* **Git & GitHub:** Version control and code hosting

**Smart Contract Structure**

The contract uses the ERC20 standard to issue loyalty tokens. Key functions include:

* mint() - Owner can issue tokens to users
* transfer() - Users can send tokens to others
* redeem() - Tokens can be burned or marked as used when redeemed

**Token Flow Diagram**

Customer Purchase --> Owner issues Tokens --> Customer uses tokens --> Redeems/Transfers

**Smart Contract Snippet**

function mint(address to, uint256 amount) public onlyOwner {

\_mint(to, amount);

}

function redeem(uint256 amount) public {

require(balanceOf(msg.sender) >= amount, "Insufficient tokens");

\_burn(msg.sender, amount);

}

These functions ensure only the contract owner can mint tokens, while users can redeem them securely.

**Deployment & Testing**

**Tools Used**

* Remix IDE for contract deployment
* MetaMask for wallet integration
* Sepolia Testnet for deployment

**Steps**

1. Deployed smart contract using Injected Provider (MetaMask)
2. Interacted with mint, transfer, and redeem functions
3. Pushed contract to GitHub for version control and transparency

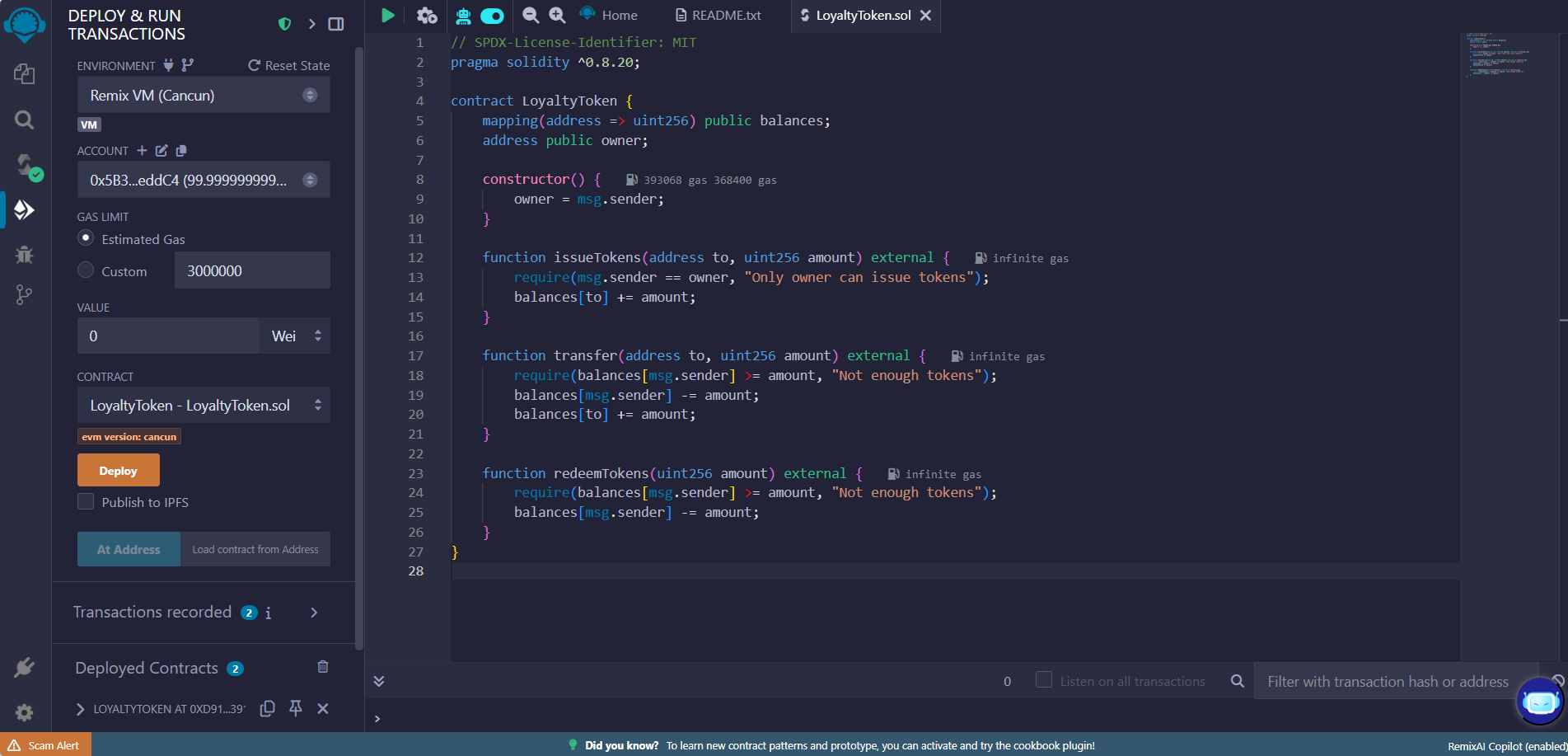
**GitHub Link**

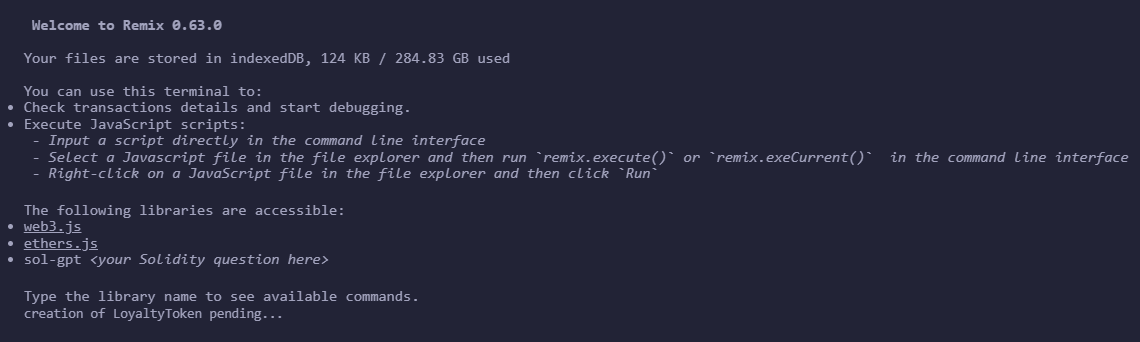
<https://github.com/aks021102/LoyaltyToken>

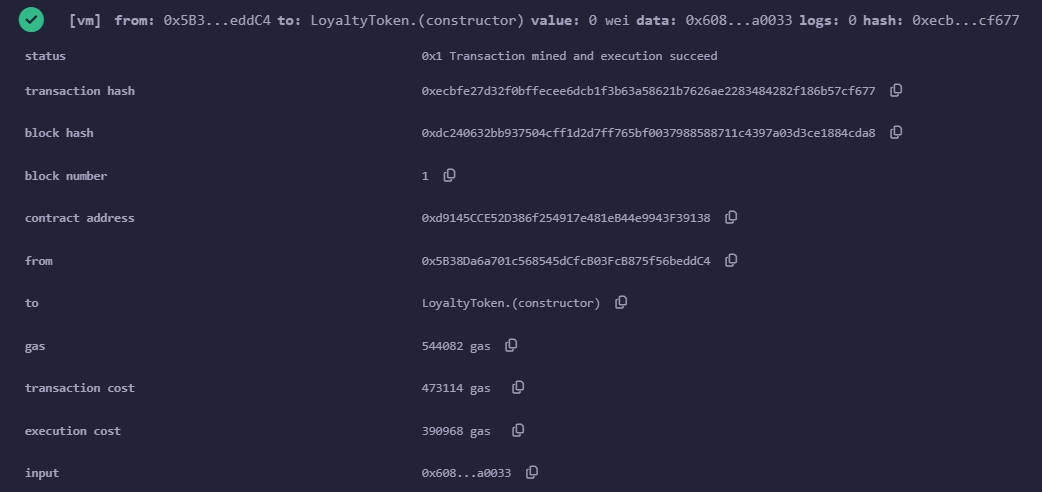
**Challenges Faced**

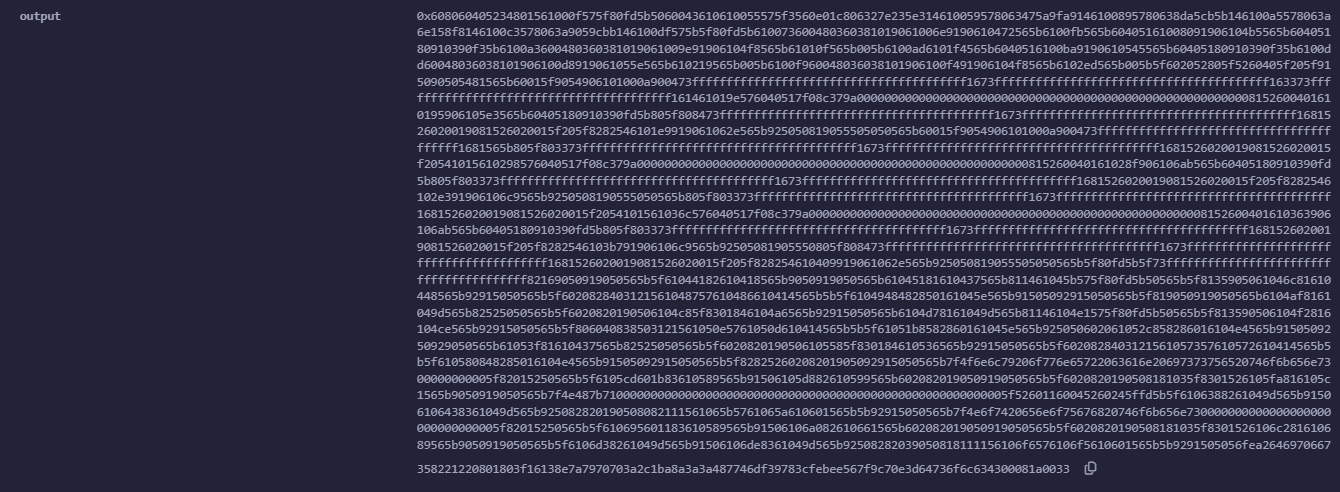
* Initial issue with not seeing MetaMask provider in Remix
* Faucet limitations on Sepolia Testnet
* Git configuration errors during first commit

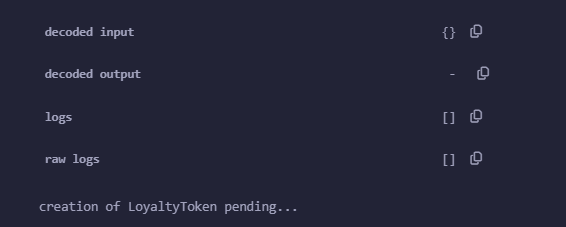
**IMPLEMENTATION DETAILS:**

****

****

****

****

****

**Conclusion & Future Scope**

This project demonstrates a functional prototype of a token-based loyalty system using Ethereum smart contracts. In future versions, features like expiry dates, dynamic reward tiers, and admin dashboards can be added to enhance real-world adoption.

**References**

1. Ethereum Documentation - <https://ethereum.org/en/developers/>
2. Solidity Docs - [https://docs.soliditylang.org](https://docs.soliditylang.org/)
3. Remix IDE - [https://remix.ethereum.org](https://remix.ethereum.org/)
4. MetaMask - [https://metamask.io](https://metamask.io/)