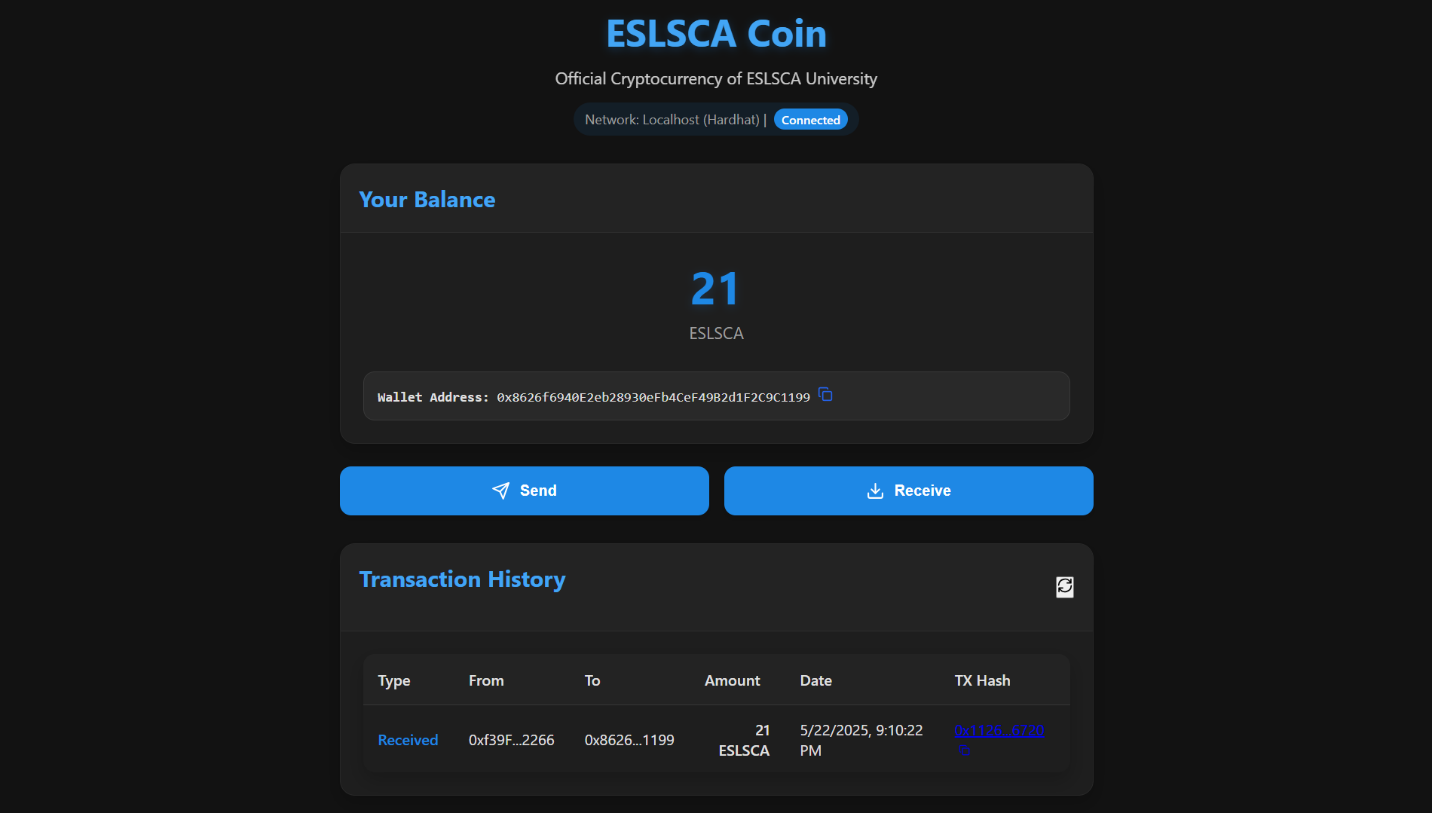
**ESLSCA Coin - Project Documentation**

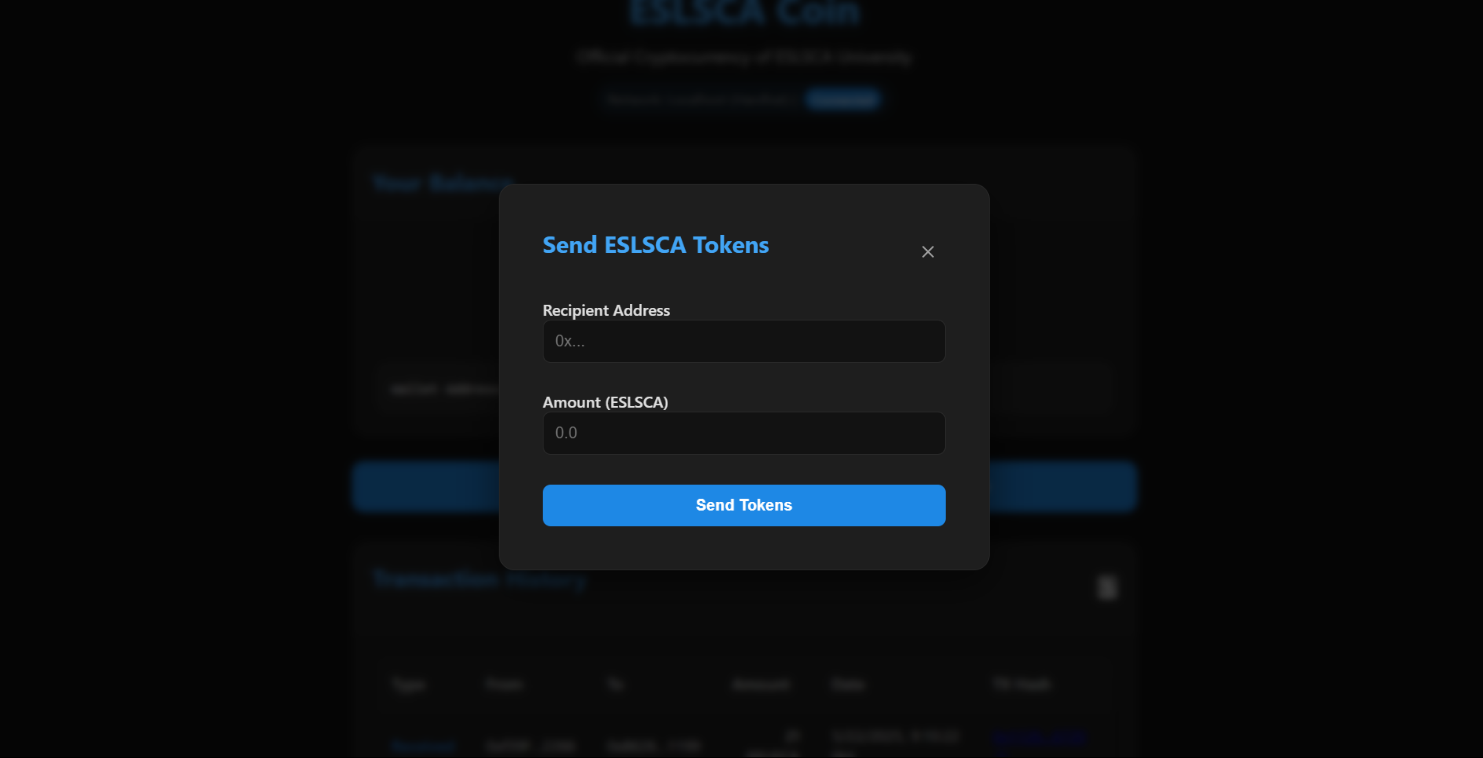
**1. Executive Summary**

ESLSCA Coin is a fully functional ERC-20 token implementation deployed on the Ethereum blockchain, specifically designed for ESLSCA University. This project demonstrates the practical application of blockchain technology in creating a digital currency that can be transferred between users, with administrative capabilities for minting new tokens.

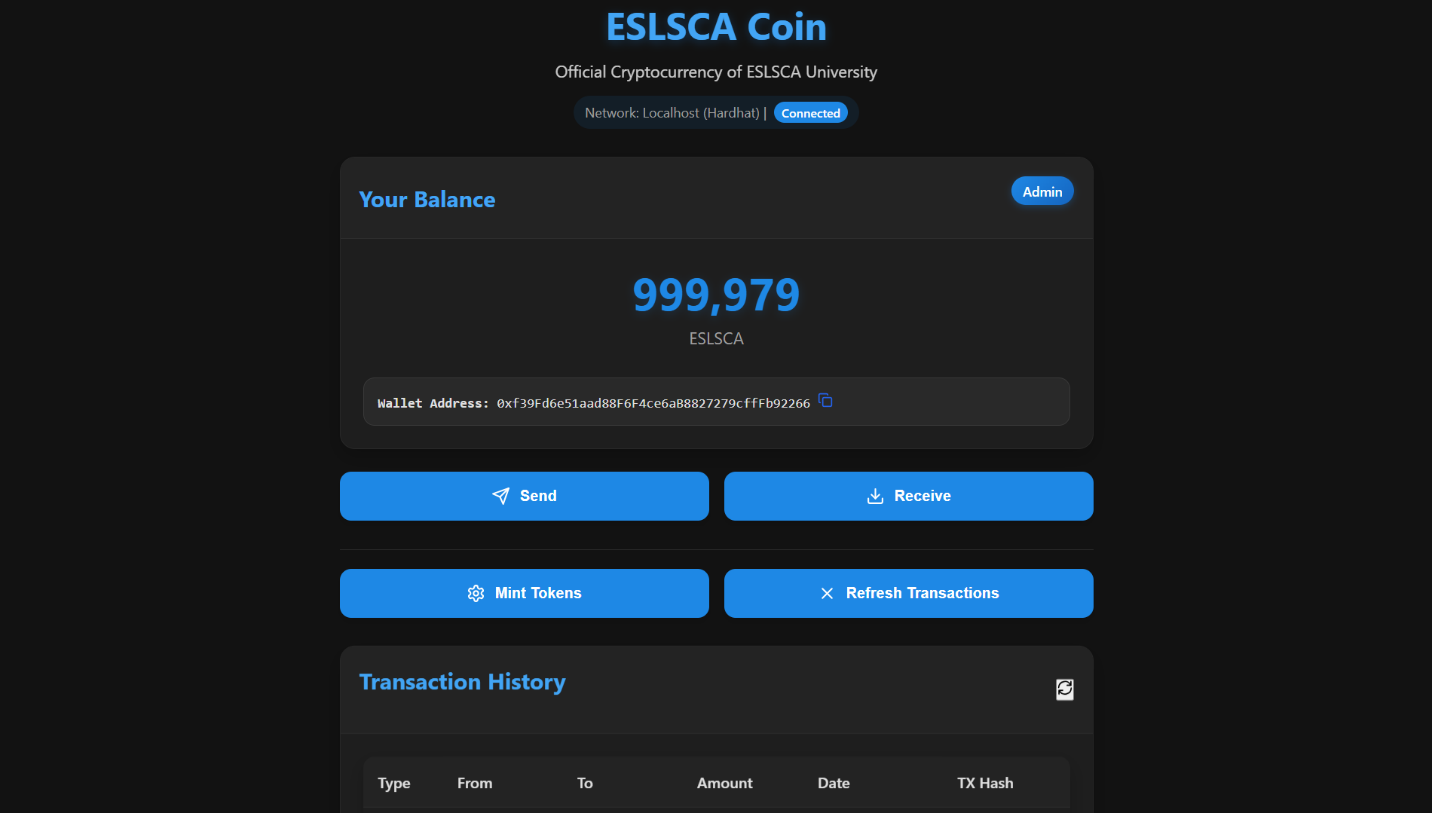
**The project includes:**

* A Solidity-based smart contract implementing the ERC-20 token standard
* A modern, responsive web interface for interacting with the token
* Comprehensive testing and deployment infrastructure
* MetaMask wallet integration



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**2. Problem Statement**

Traditional financial systems within educational institutions face several challenges:

* No change is found sometimes when buying from cafeteria, etc.
* Slow processing times for payments
* Limited transparency in financial operations
* Difficulty implementing reward systems for academic achievements

ESLSCA Coin addresses these challenges by providing a secure, transparent, and efficient digital currency specifically designed for the university ecosystem.

**3. Project Objectives**

* Create a fully functional ERC-20 token with custom features for ESLSCA University
* Develop a user-friendly interface for managing tokens and transactions
* Implement secure administrative controls for token management
* Provide transparent transaction history and token traceability
* Ensure robust testing and deployment procedures
* Demonstrate practical application of blockchain technology in an educational context

**4. Technical Architecture**

**Smart Contract (Blockchain Layer)**  
The core of ESLSCA Coin is an ERC-20 smart contract built on the Ethereum blockchain. The contract leverages OpenZeppelin's well-audited libraries to ensure security and compliance with token standards.

**Key components:**

* ESLSCACoin.sol: Main contract implementing ERC-20 functionality with added features
* Custom event emissions for enhanced tracking
* Owner-restricted minting capabilities
* Additional utility functions for contract information

**Frontend Application (User Interface Layer)**  
A React-based single-page application provides intuitive access to ESLSCA Coin functionality:

**Key components:**

* Wallet connection integration with MetaMask
* Token balance display
* Transfer functionality
* Transaction history
* Administrative interface for token minting

**Development Infrastructure**

* Hardhat: Development environment for compilation, testing, and deployment
* Ethers.js: Library for interacting with the Ethereum blockchain

**5. Smart Contract Details**

**ESLSCACoin Contract**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

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

import "@openzeppelin/contracts/access/Ownable.sol";

/\*\*

 \* @title ESLSCACoin

 \* @dev Implementation of ESLSCA Coin - A cryptocurrency for ESLSCA University

 \*/

contract ESLSCACoin is ERC20, Ownable {

    // Events

    event TokensMinted(address indexed to, uint256 amount, uint256 timestamp);

    event TransferExecuted(

        address indexed from,

        address indexed to,

        uint256 amount,

        uint256 timestamp

    );

    // Constructor

    constructor(

        string memory name,

        string memory symbol,

        uint256 initialSupply,

        address initialOwner

    ) ERC20(name, symbol) Ownable(initialOwner) {

        // Mint initial supply to the owner (initialSupply should already be in wei)

        \_mint(initialOwner, initialSupply);

        emit TokensMinted(initialOwner, initialSupply, block.timestamp);

    }

    /\*\*

     \* @dev Mint new tokens - only owner can mint

     \* @param to Address to mint tokens to

     \* @param amount Amount of tokens to mint (in wei - same units as totalSupply)

     \*/

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

        require(to != address(0), "Cannot mint to zero address");

        require(amount > 0, "Amount must be greater than 0");

        \_mint(to, amount);

        emit TokensMinted(to, amount, block.timestamp);

    }

    /\*\*

     \* @dev Override transfer to add custom event

     \*/

    function transfer(

        address to,

        uint256 amount

    ) public virtual override returns (bool) {

        address sender = \_msgSender();

        bool success = super.transfer(to, amount);

        if (success) {

            emit TransferExecuted(sender, to, amount, block.timestamp);

        }

        return success;

    }

    /\*\*

     \* @dev Override transferFrom to add custom event

     \*/

    function transferFrom(

        address from,

        address to,

        uint256 amount

    ) public virtual override returns (bool) {

        bool success = super.transferFrom(from, to, amount);

        if (success) {

            emit TransferExecuted(from, to, amount, block.timestamp);

        }

        return success;

    }

    /\*\*

     \* @dev Get contract information

     \*/

    function getContractInfo()

        public

        view

        returns (

            string memory tokenName,

            string memory tokenSymbol,

            uint256 totalTokenSupply,

            uint8 tokenDecimals,

            address contractOwner

        )

    {

        return (name(), symbol(), totalSupply(), decimals(), owner());

    }

    /\*\*

     \* @dev Get balance of an address

     \*/

    function getBalance(address account) public view returns (uint256) {

        return balanceOf(account);

    }

}

**Key Contract Features**

**Standard ERC-20 Functionality:**

* Transfer tokens between addresses
* Check balances
* Approve spending allowances

**Enhanced Events:**

* TokensMinted: Triggered when new tokens are created
* TransferExecuted: Provides additional data for token transfers

**Administrative Controls:**

* Only the owner can mint new tokens
* Security checks to prevent minting to zero address or zero amounts

**Utility Functions:**

* getContractInfo(): Returns comprehensive contract details
* getBalance(): Convenience method for checking token balances

**6. Frontend Application**

**Technology Stack**

* React.js: Frontend framework
* Ethers.js: Ethereum interaction library
* CSS3: Custom styling with responsive design

**Key Features**

**Wallet Integration:**

* Seamless connection with MetaMask
* Account detection and display
* Network validation and switching

**Token Management:**

* Balance display with proper decimal formatting
* Token sending interface with validation
* Receipt of tokens via address sharing

**Transaction History:**

* Chronological list of all transactions
* Filtering by transaction type (sent, received, minted)
* Transaction details with links to blockchain explorer

**Administrative Panel:**

* Token minting interface (owner only)
* Batch operation capabilities

**Error Handling:**

* Comprehensive error detection and user feedback
* Multiple fallback mechanisms for blockchain interactions
* Clear success and error notifications

**7. Testing Strategy**

A comprehensive testing approach ensures the reliability and security of the ESLSCA Coin implementation:

**Smart Contract Tests**

**Deployment Tests:**

* Verify correct owner assignment
* Confirm initial token supply allocation
* Validate token metadata (name, symbol, decimals)

**Transaction Tests:**

* Test token transfers between accounts
* Verify balance updates after transfers
* Ensure transactions fail with insufficient balances

**Minting Tests:**

* Confirm owner can mint new tokens
* Verify non-owners cannot mint tokens
* Validate restrictions on minting to zero address
* Test restrictions on minting zero amounts

**Event Tests:**

* Verify TokensMinted event emission
* Confirm TransferExecuted event with correct parameters

**Utility Function Tests:**

* Verify getContractInfo returns correct data
* Test getBalance functionality

**8. Deployment Process**

The deployment process is automated to ensure consistency and reliability:

**Local Development:**

* Start local Hardhat node (npm run node)
* Deploy contract to local network (npm run deploy)
* Deployment information automatically saved to frontend

**Frontend Deployment:**

* Build optimized frontend (cd frontend && npm run build)
* Serve via appropriate web hosting

**MetaMask Configuration:**

* **Network:** Localhost (for development) or appropriate Ethereum network
* **RPC URL:** http://127.0.0.1:8545 (local) or network provider
* **Chain ID:** 1337 (local) or appropriate network ID
* **Currency Symbol:** ESLSCA

**9. Security Considerations**

Several security measures have been implemented to protect the ESLSCA Coin ecosystem:

**Smart Contract Security:**

* Use of OpenZeppelin's audited contracts
* Input validation for all public functions
* Access control via Ownable pattern
* Comprehensive testing of edge cases

**Frontend Security:**

* Transaction validation before submission
* Proper error handling and user feedback
* Protection against common web vulnerabilities

**User Security:**

* MetaMask integration for secure key management
* Transaction confirmation requirements
* Clear display of transaction details before signing

**10. Future Enhancements**

The ESLSCA Coin project has several potential areas for future development:

**Integration Capabilities:**

* APIs for third-party applications
* Integration with university payment systems

**Advanced Features:**

* Staking mechanisms for rewards
* Tokens for scholarships
* Automated token distribution based on academic achievements

**Enhanced Analytics:**

* Advanced transaction reporting
* Token economics visualization

**11. Conclusion**

ESLSCA Coin demonstrates a practical implementation of blockchain technology in an educational context. By providing a secure, transparent, and efficient digital currency, it addresses multiple challenges in traditional financial systems while offering enhanced capabilities for future innovation.

The project successfully fulfills its objectives of creating a functional ERC-20 token with custom features, developing a user-friendly interface, implementing secure administrative controls, and providing transparent transaction tracking.

**12. Appendices**

**Appendix A: Installation and Setup**

# Clone the repository

git clone https://github.com/Im2rnado/eslsca-coin.git

cd eslsca-coin

# Install dependencies

npm install

cd frontend && npm install && cd ..

# Compile contracts

npm run compile

# Run tests

npm test

# Start local blockchain

npm run node

# Deploy contract (in a new terminal)

npm run deploy

# Start frontend

npm run dev

**Appendix B: User Guide**

1. **Connecting Your Wallet:**

* Install MetaMask browser extension
* Configure MetaMask for the appropriate network
* Click "Connect MetaMask" in the ESLSCA Coin application

1. **Viewing Your Balance:**

* Your ESLSCA Coin balance appears in the Balance Card
* The wallet address is displayed below the balance

1. **Sending Tokens:**

* Click the "Send" button
* Enter the recipient's address and amount
* Click "Send Tokens" to confirm

1. **Receiving Tokens:**

* Click the "Receive" button
* Share your wallet address with the sender
* Copy address with the copy button

1. **Viewing Transaction History:**

* Scroll down to the Transaction History section
* View details of past transactions
* Click "Refresh" to update the list

1. **Admin Functions (Owner only):**

* Mint new tokens via the "Mint Tokens" button
* Enter recipient address and amount
* Click "Mint Tokens" to create new tokens