**THE UNIVERSITY OF DODOMA**

**COLLEGE OF INFORMATICS AND VIRTUAL EDUCATION**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (CSE)**

**GROUP NO 03**

**PROJECT TITLE: SMART SUBSIDY FOR SMALLHOLDERS FARMERS**

|  |  |  |
| --- | --- | --- |
| S/N | REGISTRATION NUMBER | NAME |
| 1 | T22-03-14261 | NELSON FANUEL |
| 2 | T22-03-00295 | HENRY SELEMAN |
| 3 | T22-03-09461 | EUSEBIUS MATHIAS |
| 4 | T22-03-09409 | JOYCE FUMBO |
| 5 | T22-03-05056 | MASANJA MAKALANGA |
| 6 | T22-03-08101 | REMIGIUA WILBERT |

Contents

[1. Introduction 3](#_Toc199409231)

[2. Problem Statement 3](#_Toc199409234)

[3. Objectives 4](#_Toc199409240)

[4. Scope of the Project 4](#_Toc199409245)

[5. Tools and Technologies used 4](#_Toc199409251)

[6. Smart Contract Codes 5](#_Toc199409265)

[**7.** Setting Up Hardhat Node for Local Testing 7](#_Toc199409302)

[8. Running Deployment Scripts on Local Blockchain 7](#_Toc199409307)

[9. Backend Server Setup for Frontend-Blockchain Interaction 8](#_Toc199409312)

[10. Server URL for Testing and Integration 9](#_Toc199409315)

[**11.** User Registration Flow with MetaMask Integration 10](#_Toc199409318)

12. Receiving and Using Subsidy Tokens………………………………………………………………………………………………11

13. Security Considerations and Measures……………………………………………………………………………………………14

[14. Token Visibility and Use 15](#_Toc199409343)

[15. Conclusion 16](#_Toc199409349)

16. References………………………………………………………………………………………………………………………………………16

Smart subsidy system for smallholder Farmers Using Blockchain

# 1. Introduction

## The Smart Subsidy System for Smallholder Farmers leverages blockchain technology to automate and secure the distribution of agricultural subsidies. By utilizing Ethereum smart contracts and a decentralized application (DApp) interface, the system ensures transparency, minimizes fraud, and promotes inclusion. The platform includes modules for farmer registration, tokenized subsidies and real-time auditing. Subsidies play a crucial role in supporting smallholder farmers by providing financial assistance to improve agricultural productivity. However, traditional subsidy systems often suffer from inefficiencies, lack of transparency, and potential corruption.

## This project proposes a **blockchain-based smart subsidy distribution system** that ensures **fair, transparent, and automated** allocation of subsidies using **tokenized payments**. The system will identify eligible farmers, verify their details transparently, and distribute digital tokens that can be redeemed for agricultural inputs.

# **2.** Problem Statement

## Current subsidy distribution systems are plagued by:

## **Lack of transparency** in eligibility verification and fund disbursement.

## **Manual processes** prone to errors and corruption.

## **Delays in distribution** and lack of traceability.

## **Exclusion of genuine beneficiaries** due to poor data management.

# 3. Objectives

## - Ensure transparency and accountability in subsidy distribution.

## - Automate the eligibility and distribution process using smart contracts.

## - Facilitate token-based redemption for approved goods and services.

## - Provide dashboards for farmers and administrators.

# **4. Scope of the Project**

## **Farmer Registration:** Farmers register with personal and land information.

## **Token Distribution:** Eligible farmers receive digital tokens.

## **Wallet Integration**: MetaMask wallet for managing tokens.

## **Admin Panel:** For oversight, eligibility control, and token minting.

## **Transparency:** All transactions are logged on the blockchain.

# 5. Tools and Technologies used

## **Blockchain & Smart Contracts**

## **Hardhat** – Development environment for Ethereum smart contracts.

## **Solidity** – Smart contract programming language.

## **OpenZeppelin** – Standard contracts (ERC-20).

## **Backend**

## **Node.js –** Server-side JavaScript runtime.

## **Express.js** – Web server framework.

## **ethers.js** – Ethereum library for smart contract interaction.

## **MySQL** – Database for user information.

## **Frontend**

## **html** – Frontend library.

## **MetaMask** – Browser extension for wallet and transaction signing.

## **web3.js** – Blockchain interaction from frontend.

# 6. Smart Contract Codes

## **Lock.sol;** This contract could be used for**, Time locked investments**: Locking funds for future release and **Savings contracts:** Prevent premature withdrawal.

## **// SPDX-License-Identifier: UNLICENSED**

## **pragma solidity ^0.8.28;**

## **// Uncomment this line to use console.log**

## **// import "hardhat/console.sol";**

## **contract Lock {**

## **uint public unlockTime;**

## **address payable public owner;**

## **event Withdrawal(uint amount, uint when);**

## **constructor(uint \_unlockTime) payable {**

## **require(**

## **block.timestamp < \_unlockTime,**

## **"Unlock time should be in the future"**

## **);**

## **unlockTime = \_unlockTime;**

## **owner = payable(msg.sender);**

## **}**

## **function withdraw() public {**

## **// Uncomment this line, and the import of "hardhat/console.sol", to print a log in your terminal**

## **// console.log("Unlock time is %o and block timestamp is %o", unlockTime, block.timestamp);**

## **require(block.timestamp >= unlockTime, "You can't withdraw yet");**

## **require(msg.sender == owner, "You aren't the owner");**

## **emit Withdrawal(address(this).balance, block.timestamp);**

## **owner.transfer(address(this).balance);**

## **}**

## **}**

## **SubsidyToken.sol**

## This contract is a simple ERC20 token implementation using OpenZeppelin’s ERC20 contract which is a standard for creating fungible tokens on the Ethereum blockchain.

## **// SPDX-License-Identifier: MIT**

## **pragma solidity ^0.8.28;**

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

## **contract RuzukuToken is ERC20 {**

## **constructor(uint256 initialSupply) ERC20("Ruzuku", "RZK") {**

## **\_mint(msg.sender, initialSupply);**

## **}**

## **}**

# **7.** Setting Up Hardhat Node for Local Testing

## Starts a local **Ethereum network (node)** for development and testing purposes.

## Provides **test accounts with preloaded Ether,** simulating a real blockchain.

## **Purpose**: To deploy and test smart contracts locally, interact with them using scripts or a frontend, and simulate blockchain transactions without using a real network.

## 

# 8. Running Deployment Scripts on Local Blockchain

## **Purpose**:

## Deploys your **smart contracts** SubsidyToken.sol to a local blockchain.

## Allows to test and interact with the contracts before moving to a public or test network.

## 

# 9. Backend Server Setup for Frontend-Blockchain Interaction

## Runs the backend part of this project, enabling interaction with the blockchain through web APIs.

## 

# 10. Server URL for Testing and Integration

## Interaction with backend APIs, including routes for **user registration, token transactions, subsidy management,** and **blockchain integration.**

### 

# **11.** User Registration Flow with MetaMask Integration

## 1️. **Connect to MetaMask**

## The user clicks **"Connect to MetaMask"** in the frontend dashboard.

## The frontend calls **window.ethereum.request({ method: "eth\_requestAccounts" })** to prompt MetaMask to connect.

## The user approves the connection, and the frontend retrieves the **users Ethereum address**.

## 2️. **Register User**

## After connection, the user fills out the registration form.

## The frontend **sends the user wallet address** along with form data to the backend.

## The backend interacts with the **SubsidySystem smart contract’s registerFarmer function**, linking the user registration data to their wallet address.

## 3️. **Result**

## The user **Ethereum wallet** becomes their **unique identifier** on the blockchain.

## All future actions (eligibility check, token receipt) are tied to this wallet.

##  MetaMask provides the user's wallet address which is used as the unique identifier in the smart contract, Without connecting to MetaMask, you can’t register a user because the smart contract links data to the user's wallet address.

## 

## 

# 12. **Receiving and Using Subsidy Tokens**

## After a farmer successfully logs into the subsidy distribution system using their MetaMask wallet, they wait for the **admin to issue subsidy tokens.**

## 

## The system administrator has the authority to determine the eligibility of registered farmers. Once a farmer is verified to meet the eligibility criteria, the admin triggers the smart contract marking the farmer as eligible to receive tokens.

## Upon marking eligibility, the admin initiates the minting of **Ruzuku tokens (RZK)** using the **SubsidyToken smart contract**. The tokens are minted to the farmer connected wallet address. This ensures that subsidies are issued securely and transparently on the blockchain.

## 

## Farmers can view the amount of Ruzuku tokens they have received in real time on their dashboard. This is achieved by calling the balanceOf function from the **SubsidyToken** contract, which returns the token balance for the farmer’s wallet**.**

## This process ensures that farmers receive their subsidies in the form of blockchain tokens (RZK), which are securely tied to their registered wallet. The system maintains transparency by allowing farmers to verify eligibility and token balances directly from the blockchain.

## 

## Once tokens are received, farmers can use them as proof of subsidy or as payment in the system, depending on the application design. Tokens provide a **transparent, auditable, and traceable** method of distributing subsidies, ensuring fairness and accountability.

# 13. Security Considerations and Measures

## Security is a crucial aspect of the Automated Subsidy Distribution System. The system integrates multiple layers of security, including:

## **Smart Contract Security:** The contracts are written in Solidity following best practices to avoid vulnerabilities such as reentrancy, integer overflows, and improper access control.

## **Backend Security:** The Node.js backend uses secure communication protocols (HTTPS) and input validation to prevent common web vulnerabilities such as SQL injection and cross-site scripting (XSS).

## **Wallet Integration:** MetaMask integration ensures users have full control over their keys and transactions, reducing the risk of private key exposure.

## **Access Control:** Proper role-based access control is implemented to distinguish between administrators, farmers, and external entities.

## **Data Encryption:** Sensitive data, including farmer personal details and transaction records, are encrypted both in transit and at rest.

# **14. Token Visibility and Use**

## Once tokens are issued:

## **Real-Time Balance Updates:** The frontend queries the **balanceOf** function of the **SubsidyToken contract** to display the farmer’s current token balance.

## **Transparent Verification:** The system ensures transparency by allowing farmers to view token balances directly from the blockchain, eliminating manual reconciliation.

## **Token Utility:** The tokens (RZK) can be used for various purposes such as Proof of eligibility for further benefits

## 

# 15. Conclusion

## The **Automated Subsidy Distribution System** demonstrates a practical and innovative approach to addressing the challenges of traditional subsidy distribution mechanisms. By leveraging **blockchain technology,** the system ensures **transparency, security, and efficiency** in distributing financial support to farmers. The use of **smart contracts** on the **Ethereum network**, combined with **Node.js** and **MySQL** for backend operations, provides a solid technological foundation.

## This project not only modernizes agricultural financial management but also lays the groundwork for future enhancements, including advanced analytics, cross-platform support, and greater inclusivity. With continuous improvements and scaling, this system can serve as a model for transparent, efficient, and farmer-centric subsidy distribution programs worldwide.

# 16. References

## Ethereum Documentation: https://ethereum.org/en/developers/docs/

## Solidity Language Docs: https://soliditylang.org/docs/

## Node.js Documentation: https://nodejs.org/en/docs/

## MetaMask Developer Docs: https://docs.metamask.io/

## Tailwind CSS Docs: https://tailwindcss.com/docs

## MySQL Docs: https://dev.mysql.com/doc/