Farmer Joe' s mission is to create a decentralized platform that helps the food producer of everything

by decentralized financing of future crops. An interest rate of 0%. The payment of the loan is secured by

placing as collateral a maximum of 75% of its expected production (EP).

1.2 Vision

With the misfortune of the pandemic that hit the world and the imminent fall of the Gross Domestic Product

to catastrophic levels, Futuros Alimenticios proposes a proactive solution to future problems such as

reassembling and tracing supply chains, ensuring product placement at low prices. cost in terms of sales and

financing of different food production projects.

Currently there has been an oversupply of products and a future shortage, due to financial complications after

Covid-19. Supply chains have been broken and it is important to rethink them with a technology that is more

in line with today's world, its limitations and its opportunities.

AGRO Architecture:

AGRO is built on the Ethereum blockchain and utilizes smart contracts to enable secure and transparent transactions between farmers and investors. The architecture consists of three main components: the Ethereum blockchain, smart contracts, and the AGRO token.

1. Ethereum Blockchain:

The AGRO dApp operates on the Ethereum blockchain, which provides a decentralized and immutable ledger for recording transactions and storing data. By leveraging Ethereum's robust infrastructure, AGRO ensures transparency, security, and reliability in the allocation of crops.

2. Smart Contracts:

Smart contracts play a vital role in the AGRO ecosystem. They are self-executing contracts with predefined rules and conditions, ensuring the automation and integrity of transactions. AGRO smart contracts facilitate interactions between farmers and investors, governing processes such as loan allocation, crop trading, and commission payments.

3. AGRO Token:

The AGRO token is the native cryptocurrency of the AGRO dApp. It serves multiple purposes within the ecosystem, including incentivizing network participation, enabling commission payments, and providing access to certain features and functionalities.

Token Distribution:

The total supply of AGRO tokens is 34,000,000. The token distribution will be as follows:

- Developer Team Allocation: The developer team will receive a 12% allocation of the total token supply, which amounts to 4,080,000 AGRO tokens. This allocation serves as an incentive for the team's efforts in building and maintaining the AGRO dApp.

- User Allocation: The remaining 29,920,000 AGRO tokens will be allocated to users participating in the AGRO ecosystem. These tokens will be distributed through various mechanisms, such as token sales, rewards for active participation, and token mining, to ensure wider adoption and engagement.

It's important to note that the specific implementation details, token economics, and distribution mechanisms will require further consideration and refinement during the development process. The mentioned allocation percentages are subject to adjustment based on project requirements and community feedback.

By utilizing the Ethereum blockchain, implementing smart contracts, and introducing the AGRO token, the AGRO architecture aims to create a robust and inclusive ecosystem that empowers farmers and investors while revolutionizing the agricultural industry.

AGRO Functionality:

1. Loan Allocation Process:

Farmers can access loans through the AGRO dApp based on their expected total cost of production plus an additional 15% for contingency. The loan process involves the following steps:

1.1 Loan Application: Farmers submit loan applications, providing details such as their estimated cost of production, desired loan amount, and repayment terms.

1.2 Investor Evaluation: Investors review loan applications and assess the viability of the proposed agricultural projects. They consider factors such as the farmer's track record, crop selection, and market conditions.

1.3 Loan Approval: If an investor finds the loan application suitable, they can approve the loan and transfer the funds to the farmer's account. The loan is recorded in a smart contract, ensuring transparency and accountability.

1.4 Loan Repayment: Farmers repay the loan, including the principal amount and accrued interest, within the agreed-upon timeframe. The repayment process can be automated through smart contracts, ensuring timely payments and reducing administrative overhead.

2. Crop Trading Mechanism:

After the crops have gained value and the associated risks have decreased, investors can engage in crop trading within the AGRO ecosystem. The crop trading mechanism operates as follows:

2.1 Investor Price Adjustment: The initial investor, who provided the loan, can add a margin to their initial investment and set a new price for the future crop. This adjusted price reflects the improved value of the crop and compensates the investor for the risks undertaken.

2.2 Second Investor Participation: Another investor, seeking a lower risk investment opportunity, can choose to buy the crop at the adjusted price. This second investor assesses the crop's potential and determines if the investment aligns with their risk-return preferences.

2.3 Crop Transfer: Once the second investor agrees to purchase the crop at the adjusted price, a smart contract facilitates the transfer of ownership rights from the initial investor to the second investor. This transfer is recorded immutably on the blockchain, ensuring transparency and preventing disputes.

3. Data Management and Transparency:

AGRO ensures transparent and efficient management of agricultural data, benefiting both farmers and investors. Key features include:

3.1 Agro Nominal Data: AGRO maintains a comprehensive database of agro nominal data, including information on crops, farming practices, market trends, and historical performance. This data empowers investors to make informed decisions based on real-time and historical insights.

3.2 Immutable Records: All transactions, loan agreements, crop trades, and associated details are recorded on the Ethereum blockchain through smart contracts. This ensures transparency, auditability, and immutability of the data, reducing the potential for fraud or disputes.

3.3 Real-Time Updates: AGRO provides farmers and investors with real-time updates on crop performance, market conditions, and investment opportunities. This enables timely decision-making and enhances the overall efficiency of the agricultural value chain.

4. Commission Payments and AGRO Token Usage:

AGRO introduces its native cryptocurrency, AGRO, to facilitate commission payments and incentivize network participation. Key aspects include:

4.1 Commission Structure: AGRO charges a commission fee for facilitating loan allocations and crop trades within the platform. This fee can be a percentage of the loan amount or the transaction value, and it serves as a revenue source for the AGRO network.

4.2 AGRO Token Usage: Users can utilize AGRO tokens to pay for transaction fees, access premium features, participate in token staking or mining, and receive incentives for active participation within the AGRO ecosystem.

The AGRO functionality described above aims to empower farmers by providing access to loans, enable investors to capitalize on crop value appreciation, and enhance overall transparency and efficiency in the agricultural sector. The specific implementation details and user interfaces will be developed based on the project's requirements and community feedback.

pragma solidity ^0.8.0;

// Import the necessary zkSync libraries and contracts

import {ZkSync} from "./zkSync.sol";

contract AgroDapp {

    using ZkSync for ZkSync.BlockCommitment;

    // Struct to represent a loan

    struct Loan {

        address farmer;

        uint loanAmount;

        uint repaymentAmount;

        bool approved;

    }

    mapping (bytes32 => Loan) public loans;

    mapping (bytes32 => uint) public cropPrices;

    // zkSync Rollup contract address

    address public zkSyncContractAddress;

    constructor(address \_zkSyncContractAddress) {

        zkSyncContractAddress = \_zkSyncContractAddress;

    }

    function applyLoan(bytes32 loanId, uint loanAmount) external {

        Loan storage loan = loans[loanId];

        require(loan.farmer == address(0), "Loan already exists");

        // Perform necessary checks and validations

        // ...

        // Create a new loan

        loan.farmer = msg.sender;

        loan.loanAmount = loanAmount;

        loan.repaymentAmount = loanAmount \* 101 / 100; // Total repayment amount with 1% interest

        loan.approved = true;

        // Transfer loanAmount from investor to farmer using zkSync Rollup

        // ...

    }

    function adjustCropPrice(bytes32 cropId, uint newPrice) external {

        // Perform necessary checks and validations

        // ...

        cropPrices[cropId] = newPrice;

    }

    function buyCrop(bytes32 cropId) external payable {

        // Perform necessary checks and validations

        // ...

        uint price = cropPrices[cropId];

        require(msg.value >= price, "Insufficient funds");

        // Transfer ownership of the crop to the buyer

        // ...

        // Transfer payment to the seller (investor) using zkSync Rollup

        // ...

    }

    // Other functions and modifiers...

    // You would need to implement additional functions for zkSync Rollup integration,

    // such as depositing funds, withdrawing funds, and handling commitments.

    // Note: The above code is a simplified example and should not be used as-is for production.

    // It is crucial to consider security, data validation, access control, and other best practices

    // when implementing a real-world dApp.

    // Additionally, the zkSync integration requires specific zkSync libraries and contracts to be imported

    // and may require additional setup and configuration for proper functionality.

}

pragma solidity ^0.8.0;

// Import the necessary ERC-20 libraries

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

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

contract AGROToken is ERC20, Ownable {

    // Address to lock team allocation

    address public teamWallet;

    // Tokenomics parameters

    uint256 public constant TOTAL\_SUPPLY = 34000000 \* 10\*\*18; // Total token supply

    uint256 public constant TEAM\_ALLOCATION = 4080000 \* 10\*\*18; // 12% allocation for the team

    uint256 public constant LIQUIDITY\_ALLOCATION = 29920000 \* 10\*\*18; // 88% allocation for liquidity

    constructor(address \_teamWallet) ERC20("AGRO Token", "AGRO") {

        teamWallet = \_teamWallet;

        \_mint(address(this), TOTAL\_SUPPLY); // Mint the total supply to the contract

        // Transfer team allocation to the team wallet

        \_transfer(address(this), teamWallet, TEAM\_ALLOCATION);

    }

    // Function to release locked team tokens (can only be called by the owner)

    function releaseTeamTokens() external onlyOwner {

        require(block.timestamp >= releaseTime, "Tokens are locked");

        \_transfer(address(this), teamWallet, TEAM\_ALLOCATION);

    }

}

1. Farmer Interface:

- Login/Signup: Farmers can create an account or log in to the AGRO dApp using their credentials.

- Dashboard: Upon logging in, farmers are presented with a dashboard that displays key information such as their loan status, outstanding loan amount, and repayment schedule. They can also access other features and navigate through the dApp from here.

- Apply for Loan: Farmers can initiate the loan application process by providing details such as their estimated cost of production, desired loan amount, and repayment terms. They can submit the application for evaluation by investors.

- Loan Approval: Once a loan application is approved, farmers receive notifications and can review the loan terms, including the interest rate and repayment schedule. They can accept the loan offer and proceed with the loan disbursement.

- Crop Updates: Farmers can provide updates on their crop's progress, such as growth stages, expected yield, and any significant events that may affect the crop's value. This information is valuable for investors evaluating potential crop trades.

2. Investor Interface:

- Login/Signup: Investors can create an account or log in to the AGRO dApp using their credentials.

- Dashboard: After logging in, investors are presented with a dashboard displaying their portfolio, including active investments, loan details, and crop trades. They can access various sections and explore investment opportunities.

- Loan Evaluation: Investors can review loan applications submitted by farmers. They can analyze the proposed agricultural projects, considering factors such as the farmer's track record, crop selection, and market conditions. Based on their evaluation, they can approve or reject loan applications.

- Crop Trading: Investors can explore crop trading opportunities presented by farmers. They can view information about crops, including expected yields, current prices, and associated risks. Investors can adjust the price based on their risk appetite and make offers to purchase crops from other investors.

- Token Management: Investors can manage their AGRO token holdings, including viewing their balance, initiating transfers, and accessing token-related features such as staking or liquidity provision.

- Analytics and Insights: The dApp provides investors with analytics and insights on the agricultural market, including crop performance trends, market forecasts, and investment opportunities. This information helps investors make informed decisions.

These are just a few examples of the features and interfaces that the AGRO dApp could include. The actual design and user experience would depend on various factors, including the target audience, specific use cases, and the preferences of the development team.

pragma solidity ^0.8.0;

import "./IERC20.sol";

contract AGRO {

    string public constant name = "AGRO";

    string public constant symbol = "AGRO";

    uint8 public constant decimals = 18;

    uint256 public constant initialSupply = 34000000 \* (10 \*\* uint256(decimals));

    uint256 public constant projectAllocation = 12000000 \* (10 \*\* uint256(decimals));

    mapping(address => uint256) public balanceOf;

    mapping(address => mapping(address => uint256)) public allowance;

    uint256 public totalSupply;

    constructor() {

        totalSupply = initialSupply;

        balanceOf[msg.sender] = totalSupply - projectAllocation;

        balanceOf[address(this)] = projectAllocation;

    }

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);

    function transfer(address to, uint256 value) public returns (bool) {

        require(balanceOf[msg.sender] >= value, "Insufficient balance");

        balanceOf[msg.sender] -= value;

        balanceOf[to] += value;

        emit Transfer(msg.sender, to, value);

        return true;

    }

    function approve(address spender, uint256 value) public returns (bool) {

        allowance[msg.sender][spender] = value;

        emit Approval(msg.sender, spender, value);

        return true;

    }

    function transferFrom(address from, address to, uint256 value) public returns (bool) {

        require(balanceOf[from] >= value, "Insufficient balance");

        require(allowance[from][msg.sender] >= value, "Insufficient allowance");

        balanceOf[from] -= value;

        balanceOf[to] += value;

        allowance[from][msg.sender] -= value;

        emit Transfer(from, to, value);

        return true;

    }

}

contract LoanDApp {

    struct LoanRequest {

        address farmer;

        uint256 loanAmount;

        uint256 interestRate;

        bool active;

        address[] investors;

        mapping(address => uint256) investedAmount;

    }

    mapping(uint256 => LoanRequest) public loanRequests;

    uint256 public loanRequestCount;

    address public agroTokenAddress;

    address public usdcTokenAddress;

    address public usdtTokenAddress;

    mapping(address => bool) public approvedInvestors;

    event LoanRequestCreated(uint256 indexed requestId, address indexed farmer, uint256 loanAmount, uint256 interestRate);

    event LoanInvested(uint256 indexed requestId, address indexed investor, uint256 investedAmount);

    constructor(address \_agroTokenAddress, address \_usdcTokenAddress, address \_usdtTokenAddress) {

        agroTokenAddress = \_agroTokenAddress;

        usdcTokenAddress = \_usdcTokenAddress;

        usdtTokenAddress = \_usdtTokenAddress;

    }

    function createLoanRequest(uint256 \_loanAmount, uint256 \_interestRate) external {

        require(\_loanAmount > 0, "Loan amount must be greater than 0");

        require(\_interestRate > 0, "Interest rate must be greater than 0");

        IERC20 agroToken = IERC20(agroTokenAddress);

        require(agroToken.transferFrom(msg.sender, address(this), \_loanAmount), "AGRO token transfer failed");

        loanRequestCount++;

        LoanRequest storage request = loanRequests[loanRequestCount];

        request.farmer = msg.sender;

        request.loanAmount = \_loanAmount;

        request.interestRate = \_interestRate