INDEX

AGRICULTURE DOCS CHAIN

1. INTRODUCTION

1.1 PROJECT OVERVIEW

The Agriculture Docs Chain is an innovative endeavor designed to streamline and modernize the management of agriculture data through the utilization of blockchain technology. At its core, this project centers around the development of a smart contract on the Ethereum blockchain, meticulously crafted to store, query, and update essential agricultural information. The technical stack employed in this project is robust, with end users, a user friendly frontend developed in Visual Studio Code, and the integration of the Metamask wallet for seamless interaction with the Ethereum blockchain. The smart contract, meticulously constructed using Remix IDE and Ethereum, stands as the keystone of this project, facilitating the secure storage and management of agricultural data. To enhance user experience and accessibility, the frontend boasts a straightforward interface, featuring buttons for actions such as connecting the wallet, adding new product entries, updating existing data, and retrieving product details. Users can input product information, including product ID, name, description, and quantity, with ease. This project addresses the pressing need for efficient and secure agricultural data management, providing a solid foundation for future enhancements, including additional features, heightened security measures, and more intricate access control mechanisms. In practice, this "Agriculture Docs Chain " aims to revolutionize the agriculture industry by delivering a robust and user-friendly platform for data management and decision making, potentially transforming how farmers and stakeholders interact with agricultural data.

1.2 PURPOSE

The primary purpose of the Agriculture Docs Chain is to address the longstanding challenges associated with the management of agricultural data. It seeks to empower farmers and stakeholders with a cutting edge solution that leverages the Ethereum blockchain, thereby revolutionizing the way vital agricultural information is stored and accessed. By crafting a smart contract that allows for the addition, querying, and updating of data, the project aims to streamline the entire process, simplifying it for end users. This project serves to bridge the gap between intricate business problems in agriculture and the technological solutions that can effectively resolve them. It seeks to ensure data privacy, integrity, and security, promoting trust and transparency among users. Furthermore, the purpose extends to enhancing decision making in the agriculture sector by providing real time access to valuable insights, such as crop yields, weather conditions, and market trends. In essence, the "Agriculture Docs Chain" project is driven by the purpose of advancing the agricultural industry into a new era of data management. It promises to deliver a user-friendly platform that not only makes data entry and retrieval a breeze but also paves the way for additional features and stringent security measures, setting the stage for a fully functional and secure agricultural data management system.

2. EXISTING SYSTEM

2.1 EXISTING PROBLEM

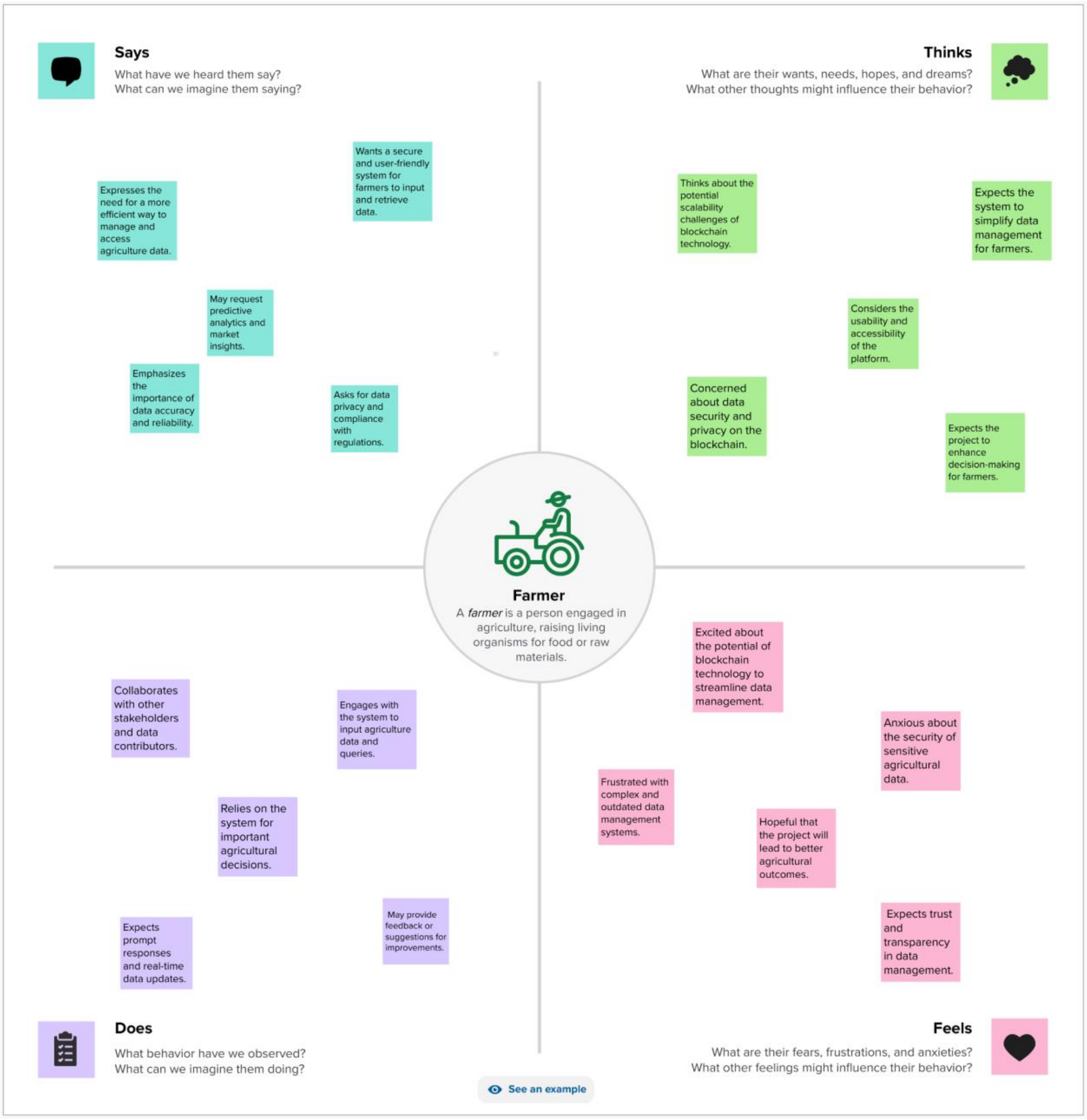
The existing problem within the agricultural sector lies in the antiquated and often inefficient methods of data management. Traditional paper based records and scattered digital documents make it challenging for farmers and stakeholders to access, update, and analyze crucial agricultural information. This not only results in time consuming processes but also increases the risk of errors, hindering informed decision making. Moreover, data security and privacy are frequently compromised in these outdated systems. The "Agriculture Docs Chain" project addresses this issue by introducing a modern, blockchain based solution that streamlines data management, enhances accessibility, and prioritizes data security, ultimately revolutionizing how agricultural data is stored and managed.

2.2 PROBLEM STATEMENT

The issue at hand revolves around the outdated and ineffective management of agricultural data, a situation that poses significant obstacles to the growth and sustainability of the agriculture sector. Currently, the prevailing systems fall short in their capacity to furnish real time data insights to both farmers and stakeholders, thereby resulting in suboptimal decision making and the potential for economic losses. The primary objective of this project is to redefine the problem statement by offering a solution that is not only crystal clear in its operation but also user friendly and highly secure for the management of agricultural data. This endeavor is centered around creating a smart contract that resides on the Ethereum blockchain, a technology renowned for its trustworthiness and transparency. This smart contract will possess the capability to efficiently handle tasks such as data addition, querying, and updates. In essence, it aims to revolutionize the management of agricultural data, addressing existing challenges in the process. Furthermore, it will significantly enhance data privacy and ensure compliance with pertinent regulations, thus heralding a transformative change in the agriculture industry. The ultimate goal is to streamline the entire process of managing agricultural data, rendering it not only easily accessible but also highly reliable and secure. In doing so, it will not only benefit the farming community but also empower stakeholders with more robust decision making tools, ultimately fostering the growth and sustainability of the agriculture sector.

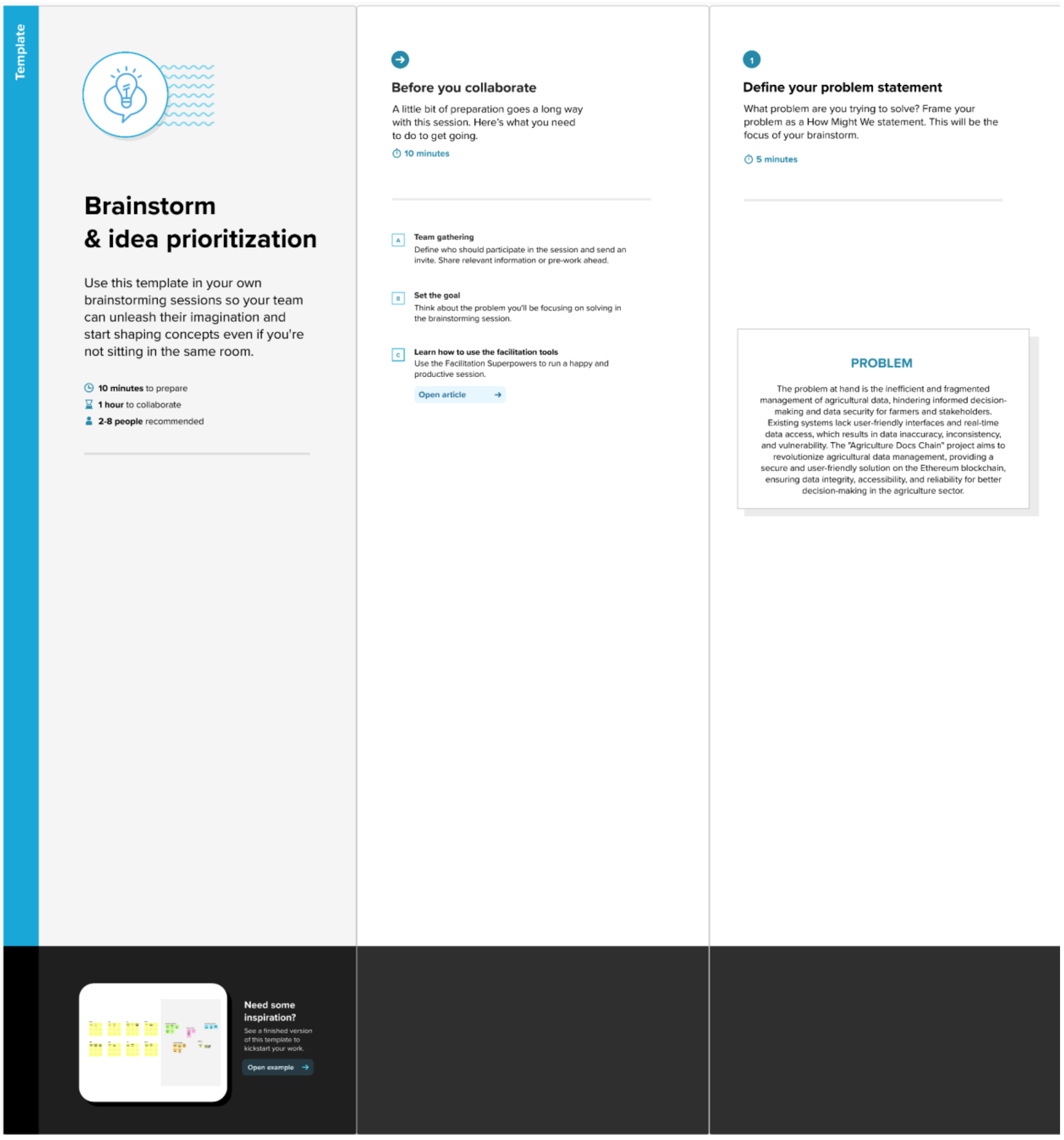
3. IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS



Says
What have we heard them say?
What can we imagine them saying?
Thinks
What are their wants, needs, hopes, and dreams?
What other thoughts might influence their behavior?
Wants a secure and user-friendly system for farmers to input and retrieve data.
Expresses the need for a more efficient way to manage and access agriculture data.
May request predictive analytics and market insights.
Emphasizes the importance of data accuracy and reliability.
Asks for data privacy and compliance with regulations.
Thinks about the potential scalability challenges of blockchain technology.
Expects the system to simplify data management for farmers.
Considers the usability and accessibility of the platform.
Concerned about data security and privacy on the blockchain.
Expects the project to enhance decision-making for farmers.
Farmer
A *farmer* is a person engaged in agriculture, raising living organisms for food or raw materials.
Collaborates with other stakeholders and data contributors.
Engages with the system to input agriculture data and queries.
Relies on the system for important agricultural decisions.
Expects prompt responses and real-time data updates.
May provide feedback or suggestions for improvements.
Excited about the potential of blockchain technology to streamline data management.
Anxious about the security of sensitive agricultural data.
Frustrated with complex and outdated data management systems.
Hopeful that the project will lead to better agricultural outcomes.
Expects trust and transparency in data management.
Does
What behavior have we observed?
What can we imagine them doing?
Feels
What are their fears, frustrations, and anxieties?
What other feelings might influence their behavior?
See an example

3.2 IDEATION AND BRAINSTORMING



Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A. **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B. **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C. **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article

1 **Define your problem statement**

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

The problem at hand is the inefficient and fragmented management of agricultural data, hindering informed decision-making and data security for farmers and stakeholders. Existing systems lack user-friendly interfaces and real-time data access, which results in data inaccuracy, inconsistency, and vulnerability. The "Agriculture Docs Chain" project aims to revolutionize agricultural data management, providing a secure and user-friendly solution on the Ethereum blockchain, ensuring data integrity, accessibility, and reliability for better decision-making in the agriculture sector.

Need some inspiration?

See a finished version of this template to kickstart your work.

Open example

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

PARAMASIVAM

Implement a user-friendly web interface for farmers to interact with the smart contract.

Integrate oracles to fetch real-world weather and market data for better decision-making.

Utilize decentralized identity systems to ensure data privacy and security.

ASWIN

Explore data encryption techniques to protect sensitive information like crop yield and farmer details.

Investigate interoperability with other blockchains or networks to enhance data sharing.

Research using NFTs to represent unique farm profiles or certifications.

SRIGANTH

Ensure regulatory compliance for agriculture data management on the blockchain.

Focus on scalability and optimize gas costs for transactions.

Explore partnerships with agricultural organizations to increase adoption.

MUTHUKUMAR

Develop a mobile app for farmers to access their data and receive notifications.

Implement an incentive mechanism using a native token for data contributors.

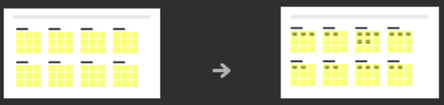
Consider integrating machine learning algorithms to provide predictive analytics.

MONIESH

Create a decentralized storage solution for large agricultural datasets.

Consider the development of a data marketplace for buying and selling agricultural insights.

Establish a governance model to involve stakeholders in decision-making.



3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

USER INTERFACE AND ACCESSIBILITY

Develop a user-friendly web interface and a mobile app for farmers.

Ensure decentralized identity and data privacy.

Explore NFTs for representing unique farm profiles.

DATA SECURITY AND PRIVACY

Implement data encryption and compliance with regulations.

Investigate decentralized storage for large datasets.

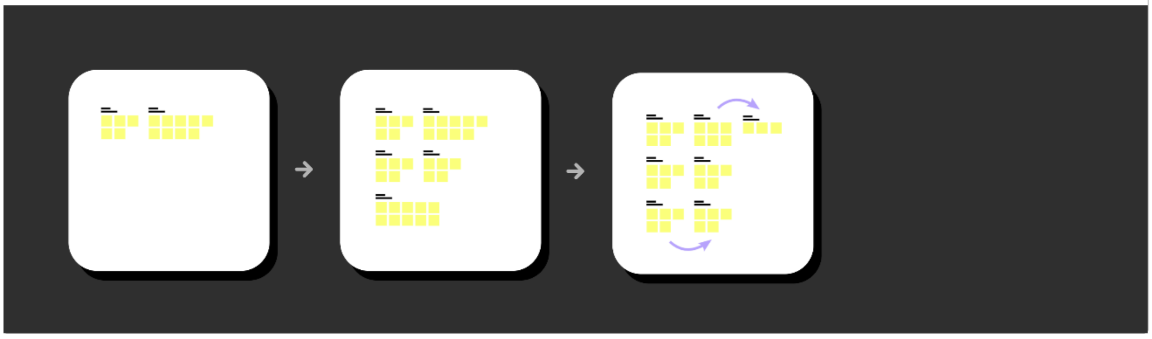
Enhance data security with oracles and encryption techniques.

DATA UTILIZATION AND INCENTIVES

Integrate machine learning for predictive analytics.

Create an incentive mechanism using a native token.

Consider a data marketplace for buying and selling insights.



4

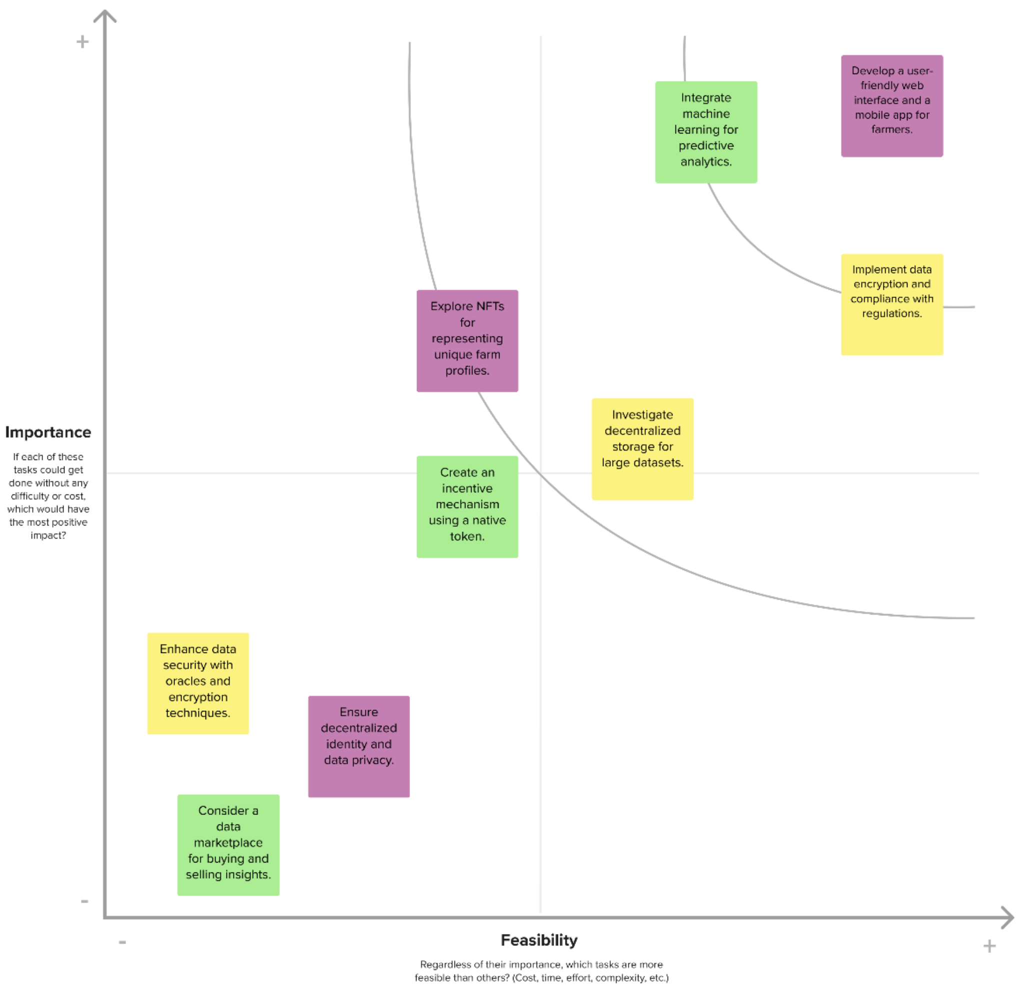
Prioritize

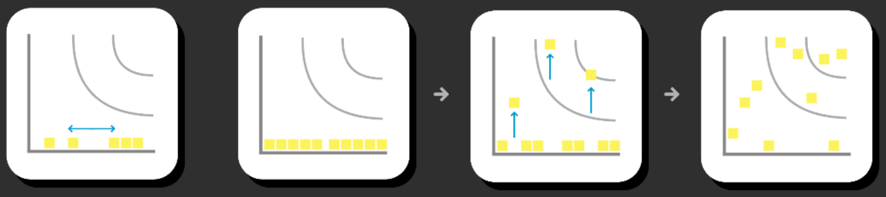
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.





4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

User Registration and Authentication: Users, primarily farmers and stakeholders, must be able to register and authenticate themselves securely. This ensures that only authorized individuals can access and interact with the system.

User Wallet Integration: The system should allow users to integrate their digital wallets, such as Metamask, to securely sign transactions and manage cryptocurrency within the application.

UserFriendly Frontend: The frontend must feature a straightforward and intuitive user interface with buttons to connect the wallet, add new product entries, update existing data, and retrieve product details. Users should easily navigate and interact with the platform.

Add Product: Users should have the ability to add agricultural products to the system. This includes input fields for product ID, product name, product description, and product quantity. The system should validate and store this data securely on the blockchain.

Update Product: Users must be able to update product information, which involves input fields for product ID, product name, product description, and product quantity. This feature ensures that the data remains accurate and uptodate.

Query Product Information: Users should be able to retrieve product details by entering a specific product ID. The system will then fetch and display the stored data, offering quick access to crucial agricultural insights.

Blockchain Smart Contract: The project's core component is the Ethereum block chainbased smart contract, written in Solidity. It should handle the addition, querying, and updating of agricultural data securely and transparently.

Data Validation: The system should incorporate data validation mechanisms to ensure that only accurate and relevant information is stored. Invalid or inconsistent data should be rejected.

Security Measures: Robust security measures, including encryption and access controls, must be in place to safeguard sensitive agricultural data. Only authorized users should have access to specific information.

Error Handling: The system should include comprehensive errorhandling procedures to provide informative feedback to users in case of data entry or transaction errors.

Compliance: The solution should adhere to relevant data privacy and regulatory requirements, ensuring that users' data is handled in a compliant and ethical manner.

Scalability and Performance: As data volumes grow, the system should be able to scale efficiently while maintaining optimal performance, ensuring that it can handle an increasing number of users and data records.

These functional requirements are critical for creating a robust and user friendly agricultural data management system that fulfills the project's objectives of secure, efficient, and accessible data handling on the blockchain.

4.2 NON FUNCTIONAL REQUIREMENT:

Performance: The system must be responsive and capable of handling a large number of simultaneous users and data transactions efficiently, with minimal latency.
Security: Robust security measures should be implemented to protect against unauthorized access, data breaches, and other security threats. Data encryption and secure authentication methods are essential.
Reliability: The system must be highly reliable, ensuring that it operates consistently without frequent downtime or data loss.
Scalability: The solution should be designed to scale as the volume of agricultural data and the number of users grow over time.

Usability: The user interfaces (web and mobile) should be intuitive, userfriendly, and accessible to individuals with varying levels of technical expertise.

Data Privacy: The system must comply with data privacy regulations and ensure that users' sensitive information is protected.

Auditability: The platform should maintain a comprehensive audit trail, recording all data transactions and interactions with the smart contract.

Disaster Recovery: A robust disaster recovery plan should be in place to ensure data recovery and system restoration in case of unexpected events or failures.

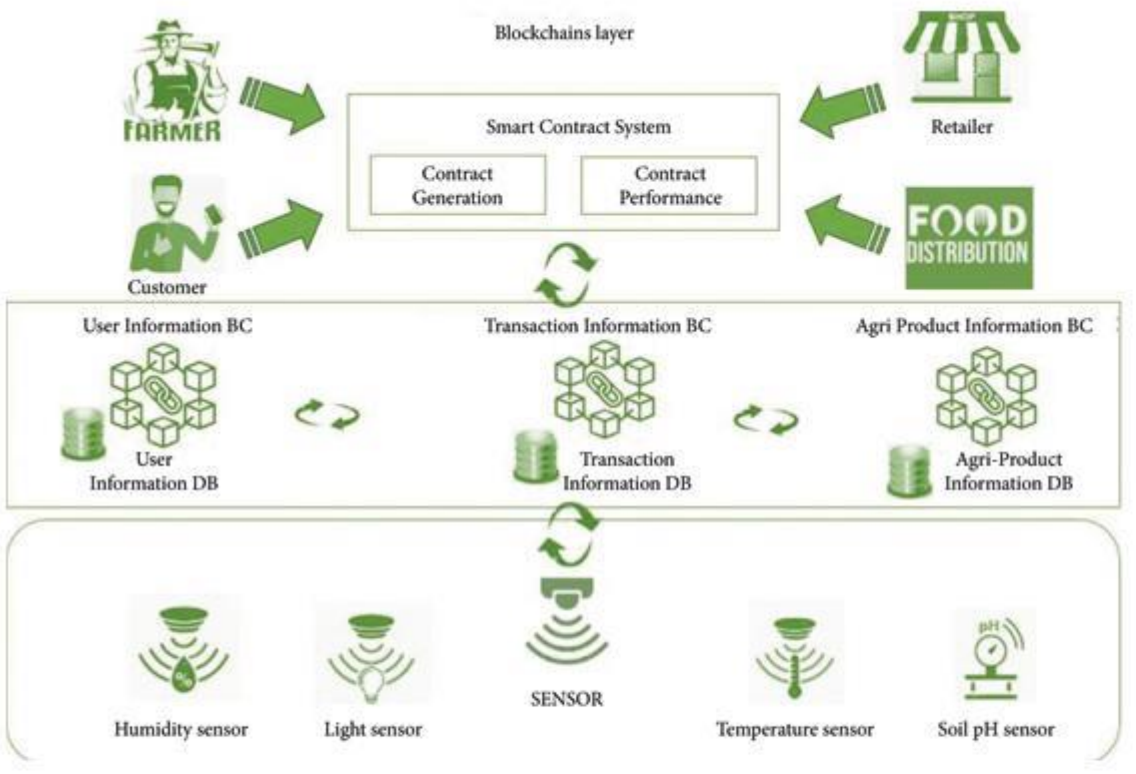
CostEffectiveness: The project should strive to maintain a cost effective approach, optimizing gas costs for transactions on the blockchain and other operational expenses.

Interoperability: The system should be designed to interoperate with other agricultural data systems, allowing for seamless data exchange and collaboration with external stakeholders.

Regulatory Compliance: Ensure that the system complies with agricultural and blockchain related regulations and standards, promoting transparency and trust among users and authority

5. PROJECT DESIGN

5.1 DATAFLOW DIAGRAM AND USER STORY



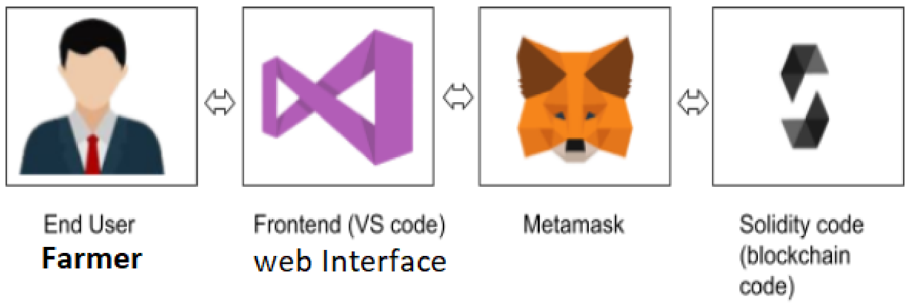
STORY 1

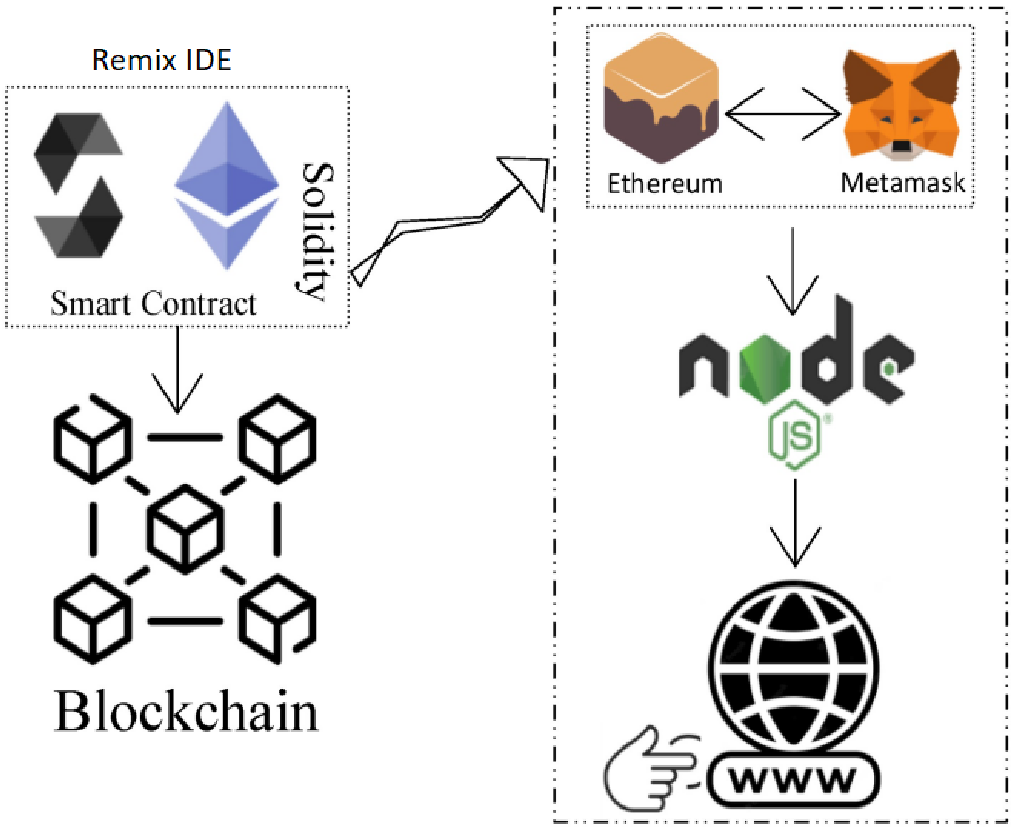
As an agriculture data manager, I want to store and manage agricultural product details on the Ethereum blockchain. I need to be able to connect my wallet through Metamask to ensure secure access. Using the frontend interface, I can add new agricultural products by specifying their unique product ID, name, description, and the quantity of the product in the project. Additionally, I should be able to update the details of existing products when necessary by providing the product's ID and the new information. This way, I can maintain accurate and up-to-date records of our agricultural projects.

STORY 2

As a user of the Agriculture Docs Chain, I want to retrieve product details from the blockchain. To do this, I enter the product's ID into the frontend interface and click the "Get Product Details" button. This feature allows me to access information about specific agricultural products that were previously added to the blockchain. It enables me to quickly and easily retrieve data about our projects and products, facilitating informed decision-making and tracking of project progress.

5.2 SOLUTION ARCHITECTURE



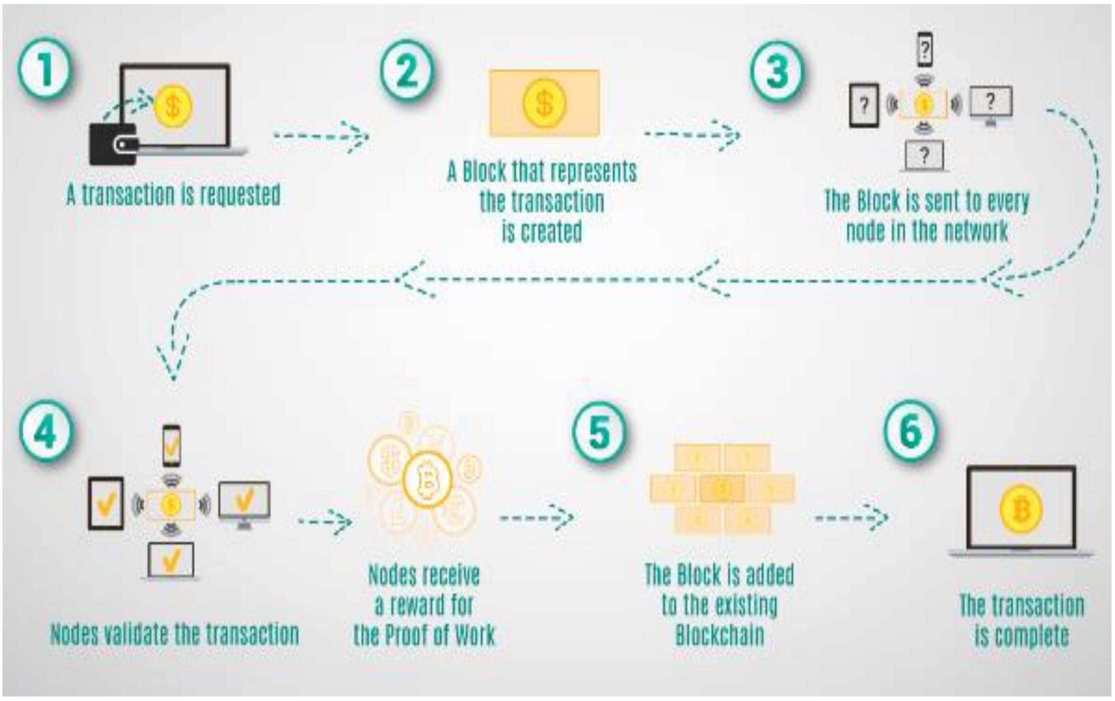


Interaction between web and the Contract

In the diagram, the end user (Farmer) is depicted at the top, using the web interface (Frontend) to access the system.
Metamask serves as the bridge between the user and the Ethereum blockchain, ensuring secure interactions with the smart contract developed in Solidity using the Remix IDE.
The Ethereum module connects to the Ethereum network, while Node.js and www (web servers) facilitate the communication between the frontend and the blockchain.
This architecture ensures data security, accessibility, and reliability, aligning with the project's goals of solving agriculture data management challenges with a robust and user-friendly solution.

6. PROJECT PLANNING & SCHEDULING

6.1 TECHNICAL ARCHITECTURE


1
A transaction is requested
2
A Block that represents the transaction is created
3
The Block is sent to every node in the network
4
Nodes validate the transaction
Nodes receive a reward for the Proof of Work
5
The Block is added to the existing Blockchain
6
The transaction is complete

6.2 Sprint Planning and Estimation

Sprint planning is a crucial part of our project development process, and it involves selecting and committing to work items from the product backlog to be completed in the upcoming sprint. Here's how we handle it:

• **Reviewing Product Backlog**: Our project team, which includes the product owner, scrum master, and development team, regularly reviews the items in the product backlog. We carefully evaluate user stories and technical tasks, taking into account the evolving needs and priorities of the project.

• **Setting Sprint Goals**: Based on the product backlog, our team establishes clear sprint goals. These goals serve as guiding principles for the team's efforts during the sprint, ensuring alignment with the broader project objectives.

• **Breaking Down User Stories**: User stories and tasks are further decomposed into smaller, actionable sub-tasks. This detailed breakdown helps in creating a comprehensive plan for the sprint, ensuring that all necessary steps are considered.

• **Estimating Work**: Our development team employs agile estimation techniques, such as story points and t-shirt sizes, to estimate the effort required for each task. These estimates provide the team with a better understanding of the scope and complexity of the work for the sprint.

• **Sprint Backlog**: The selected user stories and tasks, along with their estimates, make up the sprint backlog. This forms the basis for what the team will work on during the sprint, and it helps in managing and tracking progress effectively.

Estimation Techniques

• **Story Points**: Story points are used as a relative measure of the complexity and effort needed to complete a task. Tasks are assigned story point values based on their complexity when compared to reference tasks.

• **T-Shirt Sizes**: To provide a quick and high-level estimate of effort, tasks are categorized into t-shirt sizes, such as small, medium, and large. This approach simplifies the estimation process, especially for less complex tasks.

6.3 Sprint Delivering Schedule

Week 1: Establish the Core

- Set up the basic blockchain infrastructure.
- Implement a minimal user registration and authentication system.
- Develop a rudimentary transaction recording feature.
- Focus on fundamental security measures.

Week 2: Expand and Enhance

• Extend transaction recording to support more transaction types.

• Add basic real-time updates for transaction status.

• Enhance user authentication with multi-factor authentication.

• Begin developing a user dashboard.

Week 3: Finalize and Prepare
• Complete the user dashboard with additional features.
• Perform minimal compliance checks.
• Conduct basic testing and issue resolution.
• Create essential user support resources.

This sprint delivery schedule outlines the key objectives and tasks to be completed in each of the three weeks. It provides a clear roadmap for the development team to follow, ensuring that the project progresses smoothly and systematically.

7. CODING AND SOLUTIONING

7.1 FEATURE 1

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract AgricultureRegistry {
   struct foodProduct {
      string name;
      string description;
      uint256 quantity;
      address owner;
   }

   mapping(uint256 => foodProduct) public products;
   uint256 public productCount;

   event ProductAdded(uint256 productId, string name, string description, uint256 quantity,
address owner);
   event ProductUpdated(uint256 productId, string name, string description, uint256
quantity);
```

```
    modifier onlyOwner(uint256 _productId) {
        require(products[_productId].owner == msg.sender, "Only the owner can perform this
action");
        _;
    }

    function addProduct(uint256 ProductId, string memory _name, string memory
_description, uint256 _quantity) external {

        products[ProductId] = foodProduct(_name, _description, _quantity, msg.sender);
        productCount++;
        emit ProductAdded(productCount, _name, _description, _quantity, msg.sender);
    }

    function updateProduct(uint256 _productId, string memory _name, string memory
_description, uint256 _quantity) external onlyOwner(_productId) {
        foodProduct storage product = products[_productId];
        product.name = _name;
        product.description = _description;
        product.quantity = _quantity;
        emit ProductUpdated(_productId, _name, _description, _quantity);
    }

    function getProductDetails(uint256 _productId) external view returns (string memory
name, string memory description, uint256 quantity, address owner) {
        foodProduct memory product = products[_productId];
        return (product.name, product.description, product.quantity, product.owner);
    }
}
```
The provided Solidity smart contract, named "AgricultureRegistry," serves as a decentralized registry for tracking and managing food products within the agricultural domain. This contract features a struct named "foodProduct," which encapsulates essential information about each food product, such as its name, description, quantity, and the Ethereum address of the owner.

The contract maintains a mapping, associating each food product with a unique uint256 product identifier. Furthermore, it keeps track of the total product count. The contract includes two significant events, "ProductAdded" and "ProductUpdated," which are triggered to notify users when a new food product is added or when an existing one is updated.

7.2 FEATURE 2

FRONTEND *(JAVASCRIPT)*

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
  const [Id, setId] = useState("");
  const [Name, setName] = useState("");
  const [Desc, setDesc] = useState("");
  const [Qty, setQty] = useState("");

  const [Ids, setIds] = useState("");
  const [Names, setNames] = useState("");
  const [Descs, setDescs] = useState("");
  const [Qtys, setQtys] = useState("");
  const [Wallet, setWallet] = useState("");




  const [gId, setGIds] = useState("");
  const [Details, setDetails] = useState("");



  const handleId = (e) => {
    setId(e.target.value)
  }
```

```
const handleName = (e) => {
  setName(e.target.value)
}

const handleDesc = (e) => {
  setDesc(e.target.value)
}

const handleQty = (e) => {
  setQty(e.target.value)
}




const handleAddProduct = async () => {
  try {
    let tx = await contract.addProduct(Id.toString(), Name, Desc, Qty)
    let wait = await tx.wait()
    alert(wait.transactionHash)
    console.log(wait);
  } catch (error) {
    alert(error)
  }
}


const handleIds = (e) => {
  setIds(e.target.value)
}
```

```
const handleNames = (e) => {
  setNames(e.target.value)
}

const handleDescs = (e) => {
  setDescs(e.target.value)
}

const handleQtys = (e) => {
  setQtys(e.target.value)
}



const handleUpdate = async () => {
  try {
    let tx = await contract.updateProduct(Ids.toString(), Names, Descs, Qtys)
    let wait = await tx.wait()
    console.log(wait);
    alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}


const handleGetIds = async (e) => {
  setGIds(e.target.value)
}

const handleGetDetails = async () => {
  try {
    let tx = await contract.getProductDetails(gId.toString())
```

```
        let arr = []
        tx.map(e => {
          arr.push(e)
        })

        console.log(tx);
        setDetails(arr)
      } catch (error) {
        alert(error)
        console.log(error);
      }
    }

    const handleWallet = async () => {
      if (!window.ethereum) {
        return alert('please install metamask');
      }

      const addr = await window.ethereum.request({
        method: 'eth_requestAccounts',
      });

      setWallet(addr[0])

    }

  return (
   <div>
    <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Agriculture Registry</h1>
      {!Wallet ?

         <Button onClick={handleWallet} style={{ marginTop: "30px", marginBottom: "50px"
}}>Connect Wallet </Button>
         :
          <p style={{ width: "250px", height: "50px", margin: "auto", marginBottom: "50px",
border: '2px solid #2096f3' }}>{Wallet.slice(0, 6)}....{Wallet.slice(-6)}</p>
      }
   <Container>
    <Row>
     <Col style={{marginRight:"100px"}}>
      <div>
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleId}
type="number" placeholder="Product Id" value={Id} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleName}
type="string" placeholder="Product Name" value={Name} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleDesc}
type="string" placeholder="product description" value={Desc} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleQty}
type="number" placeholder="product quantity" value={Qty} /> <br />

       <Button onClick={handleAddProduct} style={{ marginTop: "10px" }}
variant="primary"> Add Product</Button>
      </div>
     </Col>

           <Col style={{ marginRight: "100px" }}>
             <div>
               <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleIds} type="number" placeholder="Product Id" value={Ids} /> <br />
               <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleNames} type="string" placeholder="Product Name" value={Names} />
<br />
               <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleDescs} type="string" placeholder="product description" value={Descs}
/> <br />
               <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleQtys} type="number" placeholder="product quantity" value={Qtys} />
<br />
```

```
                <Button onClick={handleUpdate} style={{ marginTop: "10px" }} variant="primary"> Update Product</Button>
              </div>
            </Col>

   </Row>
   <Row>
            <Col >
              <div style={{ margin: "auto" , marginTop:"100px"}}>
                <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleGetIds} type="number" placeholder="Enter  Id" value={gId} /><br />

                <Button onClick={handleGetDetails} style={{ marginTop: "10px" }} variant="primary">Get Product Details</Button>
                {Details ? Details?.map(e => {
                  return <p>{e.toString()}</p>
                }) : <p></p>}
              </div>
            </Col>
   </Row>
   </Container>

  </div>
 )
}

export default Home;
```

This code is a React component that represents a user interface for an "Agriculture Registry" application. The application seems to interact with a blockchain, likely Ethereum, to perform various actions related to agricultural product registration and retrieval. Here's a brief explanation of what the code provides:

State Management: The component uses the useState hook to manage various states, including product ID, name, description, quantity, a list of product details, and the user's wallet address after connecting to a wallet provider like MetaMask.

Adding Products: Users can input product details, including ID, name, description, and quantity. After entering these details and clicking the "Add Product" button, the code attempts to add this product to the blockchain by calling a contract.addProduct function. If successful, it displays a transaction hash.

Updating Products: Users can also update product details by providing a product ID, name, description, and quantity. Clicking the "Update Product" button attempts to update the product's information in a similar manner to adding.

Fetching Product Details: Users can enter a product ID and click the "Get Product Details" button. The code calls a contract.getProductDetails function, likely fetching product details from the blockchain. It then displays the retrieved details in the UI.

Wallet Connection: Before using the application, users are required to connect their Ethereum wallet (e.g., MetaMask) by clicking the "Connect Wallet" button. Once connected, the wallet's address is displayed on the screen.

React Bootstrap: The component uses the React Bootstrap library for styling and UI components like buttons, input fields, and containers.

Overall, this code provides a basic front-end for interacting with a blockchain-based agriculture registry, allowing users to add, update, and retrieve product details while also managing wallet connections.

7. PERFORMANCE TESTING

8.1 PERFORMANCE METRICS

Transaction Throughput: Measure the number of transactions the system can handle per second. High throughput is essential for accommodating a large number of users and data interactions simultaneously.

Latency: Track the time it takes for a transaction to be confirmed and recorded on the blockchain. Lower latency ensures a more responsive system**.**

Error Rates: Monitor the frequency of transaction failures or errors. Reducing error rates is crucial for ensuring data accuracy and system reliability.

Scalability: Evaluate how the system scales as the volume of data and the number of users increase. Ensure that the system can handle growth without significant performance degradation.

Resource Utilization: Assess the system's use of computational resources, memory, and network bandwidth. Efficient resource utilization helps optimize costs and maintain performance.

Gas Costs: Keep track of gas costs associated with Ethereum transactions. Optimizing gas costs is essential for cost-effective operation on the blockchain.

Response Time: Measure the time it takes for the frontend to respond to user actions. A lower response time leads to a more satisfying user experience.

Uptime and Availability: Calculate the percentage of time the system is available and operational. High uptime ensures uninterrupted service for users.

Data Retrieval Speed: Evaluate how quickly the system can retrieve and display agricultural data to users. Faster data retrieval contributes to informed decision-making.

Data Storage Efficiency: Analyze how efficiently the system stores data on the blockchain. Efficient data storage minimizes blockchain bloat and optimizes costs.

Security Audits and Vulnerabilities: Regularly conduct security audits and penetration testing to identify and address vulnerabilities, ensuring the safety of user data.

Compliance: Monitor compliance with data privacy and regulatory requirements. Ensure that user data is handled ethically and in accordance with relevant regulations.

User Adoption and Engagement: Track user adoption rates and user engagement with the system. High adoption and engagement indicate the system's value to users.

Feedback and Support Response Time: Measure the time it takes to respond to user feedback and support requests. Quick responses enhance user satisfaction and trust.

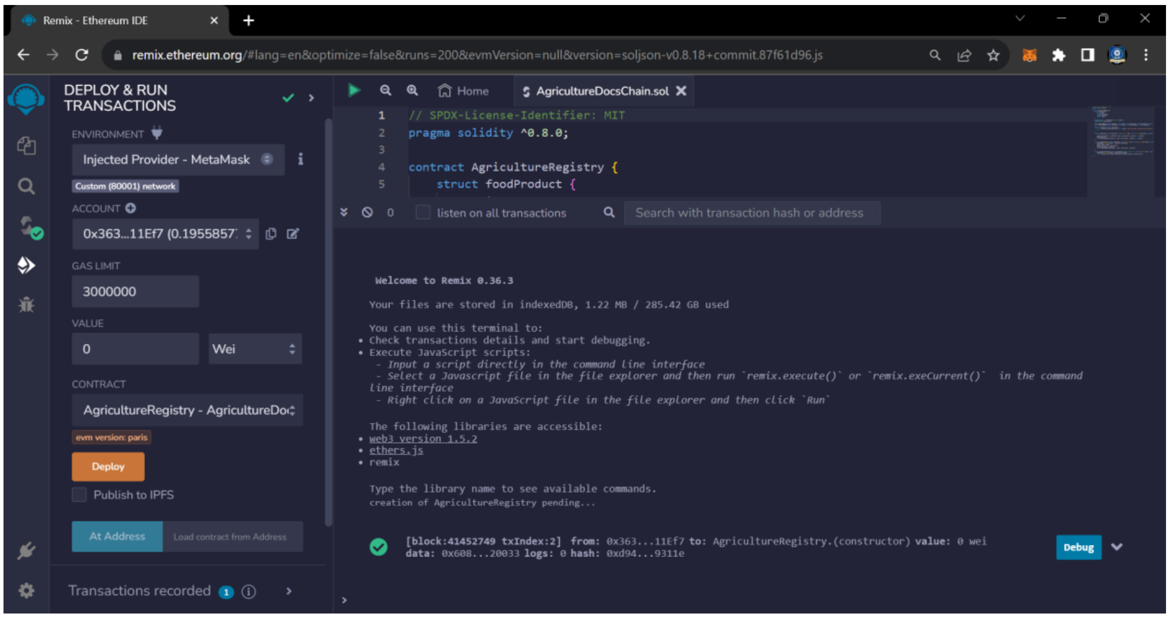
Documentation Completeness: Assess the comprehensiveness of documentation for users and developers. Well-documented systems facilitate efficient onboarding and troubleshooting.

By regularly monitoring these performance metrics, the "Agriculture Docs Chain" project can ensure that it operates effectively, efficiently, and securely while meeting user needs and expectations.

9.RESULTS

9.1 OUTPUT SCREENSHOTS

CREATING A SMART CONTRACT:

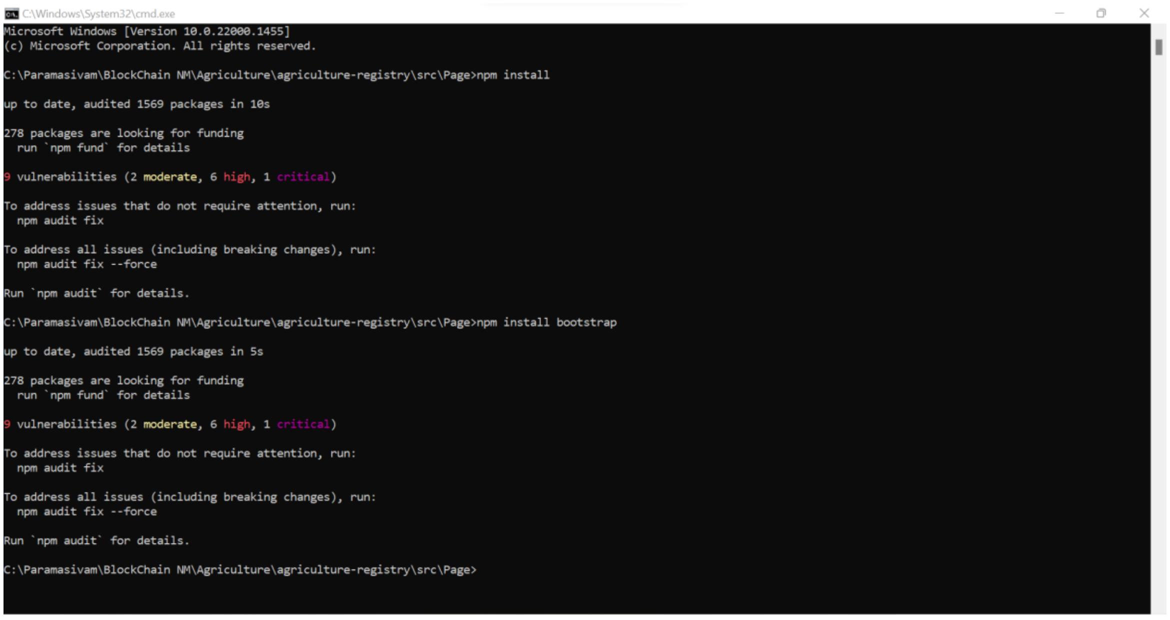

Remix - Ethereum IDE
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
DEPLOY & RUN TRANSACTIONS
Home
AgricultureDocsChain.sol
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 1
listen on all transactions
Search with transaction hash or address
Debug

ENVIRONMENT: Injected Provider - MetaMask (Custom (80001) network)
ACCOUNT: 0x363...11Ef7 (0.1955857
GAS LIMIT: 3000000
VALUE: 0 Wei
CONTRACT: AgricultureRegistry - AgricultureDoc (evm version: paris)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract AgricultureRegistry {
    struct foodProduct {
```

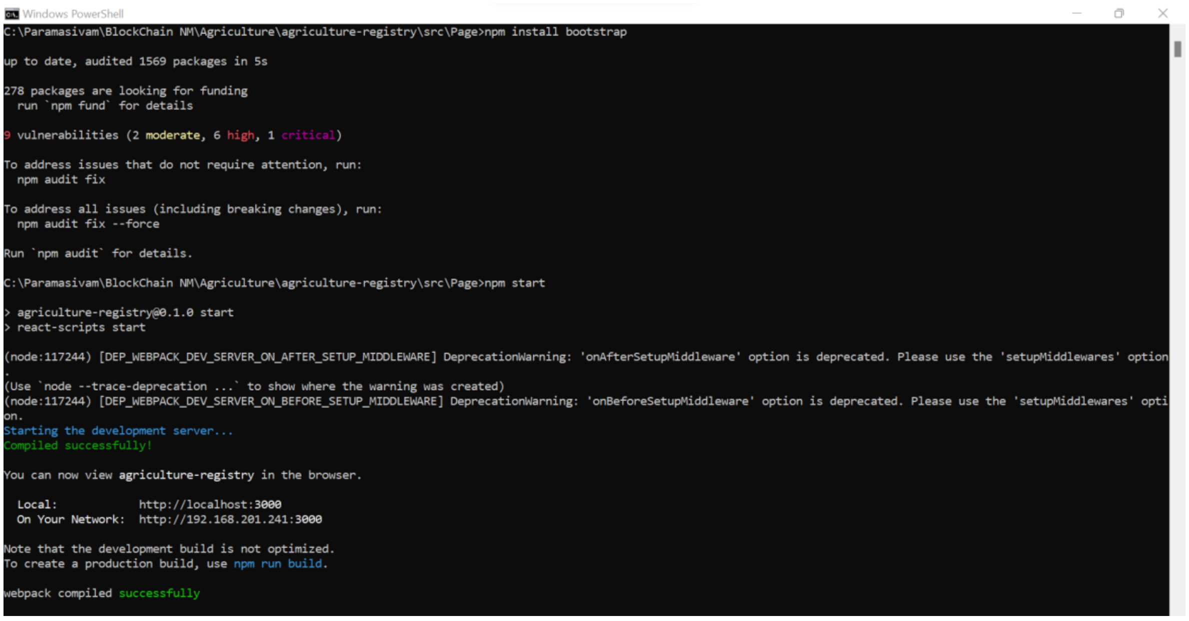
```
 Welcome to Remix 0.36.3

Your files are stored in indexedDB, 1.22 MB / 285.42 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command
line interface
  - Right click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• web3 version 1.5.2
• ethers.js
• remix

Type the library name to see available commands.
creation of AgricultureRegistry pending...

[block:41452749 txIndex:2] from: 0x363...11Ef7 to: AgricultureRegistry.(constructor) value: 0 wei
data: 0x608...20033 logs: 0 hash: 0xd94...9311e
```

INSTALLING DEPENDENCY:



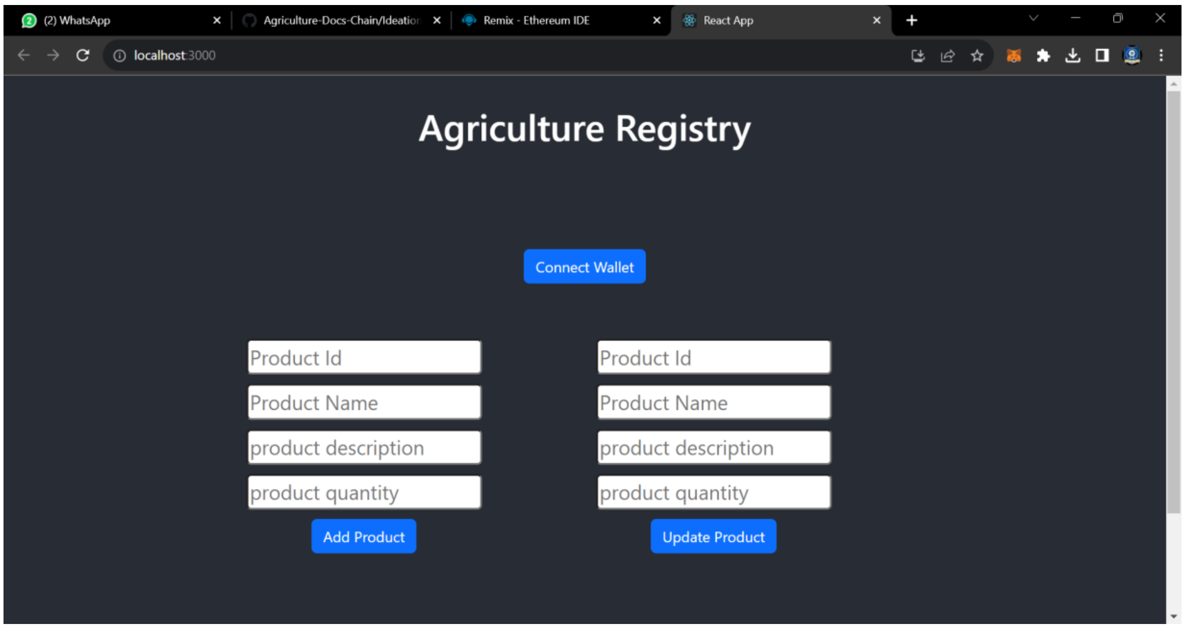
```
Microsoft Windows [Version 10.0.22000.1455]
(c) Microsoft Corporation. All rights reserved.

C:\Paramasivam\BlockChain NM\Agriculture\agriculture-registry\src\Page>npm install

up to date, audited 1569 packages in 10s

278 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Paramasivam\BlockChain NM\Agriculture\agriculture-registry\src\Page>npm install bootstrap

up to date, audited 1569 packages in 5s

278 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Paramasivam\BlockChain NM\Agriculture\agriculture-registry\src\Page>
```

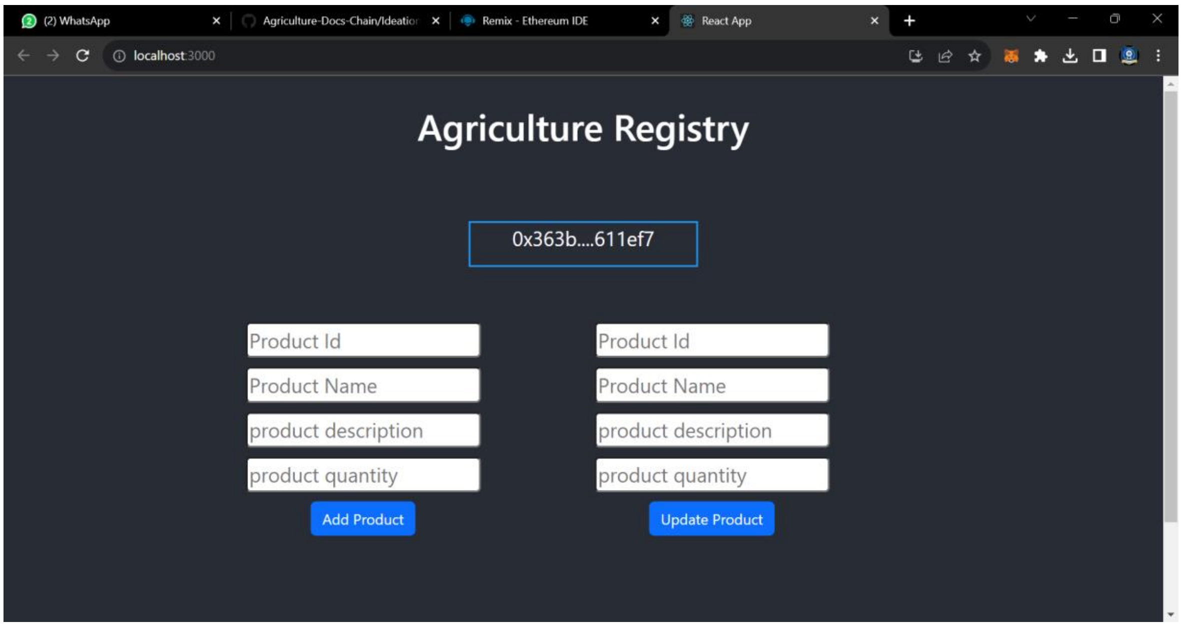
HOSTING THE SITE:



```
Windows PowerShell
C:\Paramasivam\BlockChain NM\Agriculture\agriculture-registry\src\Page>npm install bootstrap

up to date, audited 1569 packages in 5s

278 packages are looking for funding
  run `npm fund` for details

9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Paramasivam\BlockChain NM\Agriculture\agriculture-registry\src\Page>npm start

> agriculture-registry@0.1.0 start
> react-scripts start

(node:117244) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option
.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:117244) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' opti
on.
Starting the development server...
Compiled successfully!

You can now view agriculture-registry in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.201.241:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

OUTPUT SCREENSHOT:

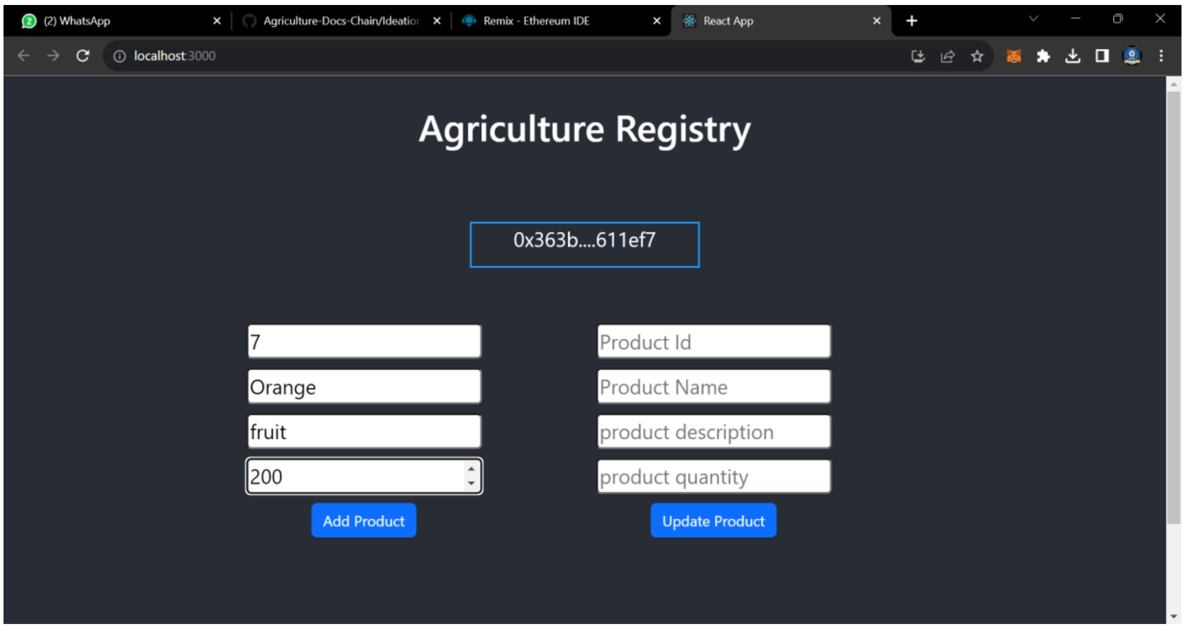




Agriculture Registry

0x363b....611ef7

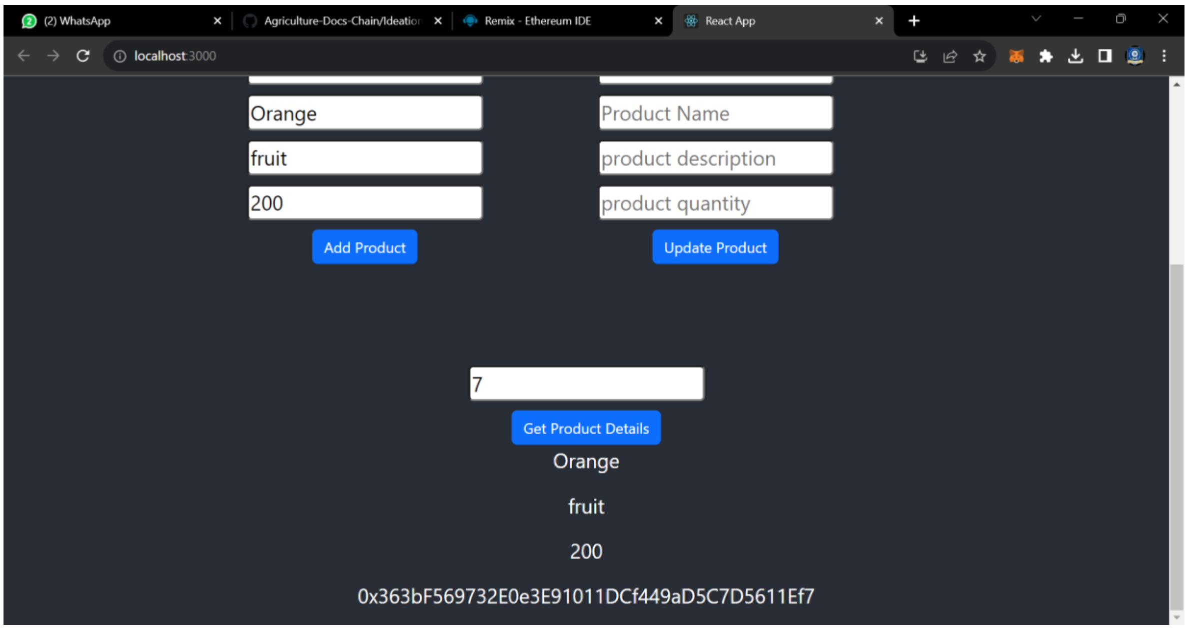
Product Id	Product Id
Product Name	Product Name
product description	product description
product quantity	product quantity
Add Product	Update Product



Agriculture Registry

0x363b....611ef7

7	Product Id
Orange	Product Name
fruit	product description
200	product quantity
Add Product	Update Product



Orange | Product Name
fruit | product description
200 | product quantity

Add Product | Update Product

7

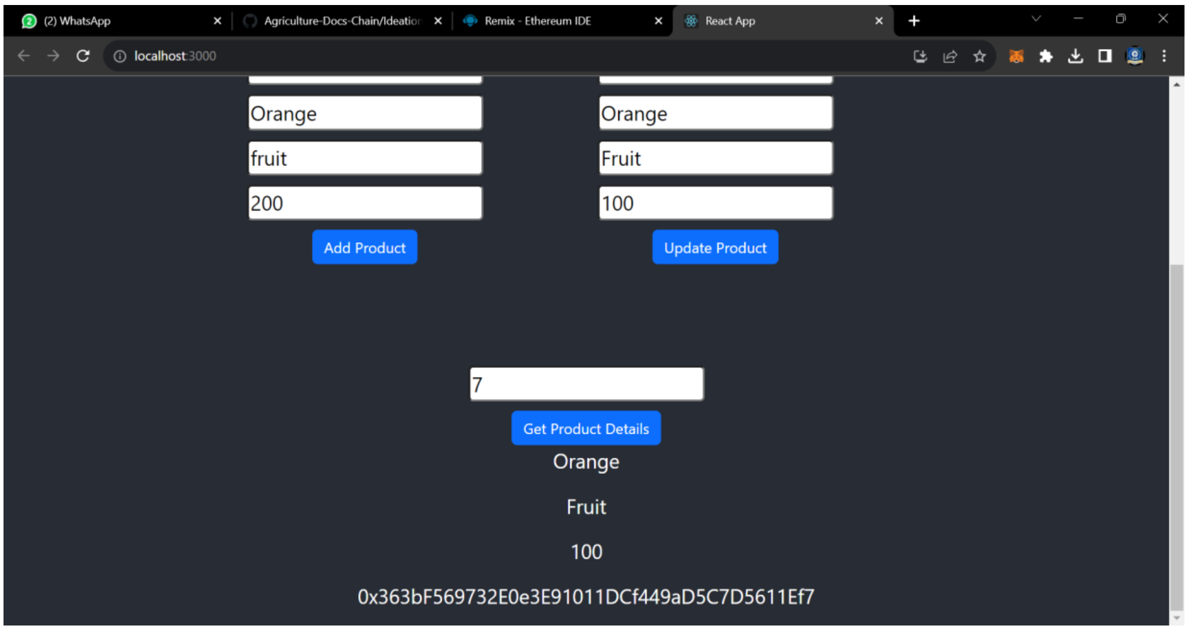
Get Product Details

Orange

fruit

200

0x363bF569732E0e3E91011DCf449aD5C7D5611Ef7



Orange | Orange
fruit | Fruit
200 | 100

Add Product | Update Product

7

Get Product Details

Orange

Fruit

100

0x363bF569732E0e3E91011DCf449aD5C7D5611Ef7

10. ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

Efficient Data Management: The project streamlines the management of agricultural data, making it more efficient for farmers and stakeholders. It offers a centralized platform for data entry, retrieval, and updates.

Real-Time Data Access: Users gain access to real-time data on crop yields, weather conditions, and market trends, enabling them to make informed decisions quickly.

Security: The use of blockchain technology and robust security measures enhances data security and privacy, reducing the risk of data breaches.

Transparency and Trust: Data stored on the blockchain is transparent and tamper-proof, enhancing trust among users and stakeholders.

User-Friendly Interface: The user interface is intuitive and user-friendly, making it accessible to individuals with varying levels of technical expertise.

Compliance: The system ensures data compliance with relevant regulations and standards, promoting ethical data handling.

Scalability: The system is designed to scale efficiently, accommodating a growing number of users and data records

10.2 DISADVANTAGES

Initial Setup Complexity: Implementing the system and configuring smart contracts may be complex, requiring technical expertise during the initial setup.

Gas Costs: Transactions on the Ethereum blockchain come with gas costs, which can be a disadvantage for frequent and small transactions, affecting cost-effectiveness.

Dependency on Blockchain: The system is reliant on the Ethereum blockchain's performance and network stability, which can be affected by network congestion and scalability issues.

User Training: Users may require training to understand how to use digital wallets and interact with blockchain-based systems.

Data Validation: Ensuring accurate and consistent data input may require additional effort to implement and maintain validation mechanisms.

Technical Support: Technical issues and user queries may require responsive technical support, which can be resource-intensive.

Regulatory Changes: Changes in blockchain and agricultural regulations may require updates and adjustments to the system to remain compliant.

11. CONCLUSION

In conclusion, the "Agriculture Docs Chain" project represents a pivotal step forward in transforming the landscape of agricultural data management. By leveraging the power of blockchain technology, this project addresses longstanding challenges and inefficiencies in the agriculture sector, offering a range of benefits to both farmers and stakeholders.

The advantages of the system are manifold. It streamlines the traditionally cumbersome process of data management, providing a centralized platform for adding, querying, and updating agricultural data. Real-time access to critical information, such as crop yields and market trends, empowers users with timely insights for making informed decisions. The project places paramount emphasis on security and transparency, utilizing blockchain to ensure tamper-proof data storage and robust data privacy measures.

However, the project does come with its set of complexities and challenges, including the initial setup intricacies and the issue of gas costs for Ethereum transactions. Yet, with a well-structured technical framework and continuous development efforts, these challenges can be effectively managed.

In the end, the "Agriculture Docs Chain" project has the potential to revolutionize the agriculture industry, offering an efficient, secure, and user-friendly solution for data management. It holds the promise of optimizing productivity, fostering trust, and enhancing data-driven decision-making in agriculture, thereby benefiting farmers, agribusinesses, and stakeholders alike. As the system continues to evolve, it promises to be a beacon of progress in the agriculture sector, driving it towards a more prosperous and sustainable future.

12. FUTURE SCOPE

The "Agriculture Docs Chain" project has significant future scope and potential for further development and expansion. Here are detailed future possibilities and areas of growth:

Integration with IoT and Sensors: The project can be extended to integrate with Internet of Things (IoT) devices and sensors placed on farms. This would allow for real-time data collection, such as soil moisture levels, temperature, and humidity. The smart contract could automatically receive and record this data, providing comprehensive insights to farmers.

Advanced Data Analytics: Implementing advanced data analytics and machine learning algorithms can help in the prediction and optimization of agricultural practices. Farmers can receive recommendations for crop planting, irrigation schedules, and pest control based on historical data.

Supply Chain Tracking: Extend the system to track the entire agricultural supply chain, from farm to table. Blockchain can be used to record and verify the origin and quality of products, enhancing food safety and traceability.

Smart Contracts for Agreements: Create smart contracts for agreements and contracts between farmers and buyers. These contracts can automatically execute payments when predefined conditions are met, enhancing trust in business transactions.

Mobile Applications: Develop dedicated mobile applications for a more seamless and accessible user experience, allowing farmers to manage their agricultural data and interact with the blockchain system on the go.

Marketplace Integration: Integrate a marketplace where farmers can sell their agricultural products directly to consumers or businesses. Blockchain can ensure transparent and secure transactions.

Multi-Blockchain Support: Consider supporting multiple blockchains or layer-2 solutions to address scalability issues and provide users with more options for interacting with the system.

Decentralized Identity: Implement decentralized identity solutions to further enhance user privacy and control over their personal information while maintaining compliance with data protection regulations.

Cross-Border Expansion: Explore opportunities to expand the project to support agriculture in different regions or even internationally. This could involve customizing the system to meet specific agricultural needs and regulations in different areas.

Data Monetization: Enable users to monetize their agricultural data by providing it to researchers, agribusinesses, or other stakeholders. This would allow farmers to benefit financially from their data contributions.

Environmental Impact Monitoring: Utilize the system to monitor and track the environmental impact of farming practices, such as water and pesticide usage. This information can be valuable for sustainability efforts and compliance with environmental regulations.

Collaboration with AgTech Companies: Collaborate with AgTech companies to enhance the project's capabilities and expand its reach. Partnerships can bring in additional resources, expertise, and industry connections.

As the agriculture industry continues to evolve and embrace digital solutions, the "Agriculture Docs Chain" project holds the potential to play a pivotal role in its transformation. By staying adaptable and responsive to emerging technologies and industry trends, the project can remain at the forefront of agricultural innovation and contribute to the sustainable growth of the sector.

13. APPENDIX

SOURCE CODE

agricultureOnBlockchain.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract AgricultureRegistry {
    struct foodProduct {
        string name;
        string description;
        uint256 quantity;
        address owner;
    }

    mapping(uint256 => foodProduct) public products;
    uint256 public productCount;

    event ProductAdded(uint256 productId, string name, string description, uint256
quantity, address owner);
    event ProductUpdated(uint256 productId, string name, string description, uint256
quantity);

    modifier onlyOwner(uint256 _productId) {
        require(products[_productId].owner == msg.sender, "Only the owner can perform
this action");
        _;
    }
    function addProduct(uint256 ProductId, string memory _name, string memory _description, uint256 _quantity) external {

        products[ProductId] = foodProduct(_name, _description, _quantity, msg.sender);
        productCount++;
        emit ProductAdded(productCount, _name, _description, _quantity, msg.sender);
    }

    function updateProduct(uint256 _productId, string memory _name, string memory _description, uint256 _quantity) external onlyOwner(_productId) {
        foodProduct storage product = products[_productId];
        product.name = _name;
        product.description = _description;
        product.quantity = _quantity;
        emit ProductUpdated(_productId, _name, _description, _quantity);
    }

    function getProductDetails(uint256 _productId) external view returns (string memory name, string memory description, uint256 quantity, address owner) {
        foodProduct memory product = products[_productId];
        return (product.name, product.description, product.quantity, product.owner);
    }
}
```

connector.js

```
const { ethers } = require("ethers");

const abi = [
 {
  "anonymous": false,
  "inputs": [
   {
    "indexed": false,
    "internalType": "uint256",
    "name": "productId",
    "type": "uint256"
   },
   {
    "indexed": false,
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
    "indexed": false,
    "internalType": "string",
    "name": "description",
    "type": "string"
   },
   {
    "indexed": false,
    "internalType": "uint256",
    "name": "quantity",
    "type": "uint256"
   },
   {
    "indexed": false,
    "internalType": "address",
    "name": "owner",
    "type": "address"
   }
  ],
  "name": "ProductAdded",
  "type": "event"
 },
 {
  "anonymous": false,
 "inputs": [
  {
   "indexed": false,
   "internalType": "uint256",
   "name": "productId",
   "type": "uint256"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "description",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "uint256",
   "name": "quantity",
   "type": "uint256"
  }
 ],
 "name": "ProductUpdated",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "ProductId",
      "type": "uint256"
     },
     {
      "internalType": "string",
      "name": "_name",
      "type": "string"
     },
     {
      "internalType": "string",
      "name": "_description",
      "type": "string"
     },
     {
      "internalType": "uint256",
      "name": "_quantity",
      "type": "uint256"
     }
    ],
    "name": "addProduct",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
   },
   {
    "inputs": [
     {
      "internalType": "uint256",
      "name": "_productId",
      "type": "uint256"
     }
    ],
    "name": "getProductDetails",
    "outputs": [
     {
```

```
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "description",
    "type": "string"
   },
   {
    "internalType": "uint256",
    "name": "quantity",
    "type": "uint256"
   },
   {
    "internalType": "address",
    "name": "owner",
    "type": "address"
   }
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [],
  "name": "productCount",
  "outputs": [
   {
    "internalType": "uint256",
    "name": "",
    "type": "uint256"
   }
  ],
  "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "uint256",
   "name": "",
   "type": "uint256"
  }
 ],
 "name": "products",
 "outputs": [
  {
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "description",
   "type": "string"
  },
  {
   "internalType": "uint256",
   "name": "quantity",
   "type": "uint256"
  },
  {
   "internalType": "address",
   "name": "owner",
   "type": "address"
  }
 ],
 "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [
   {
    "internalType": "uint256",
    "name": "_productId",
    "type": "uint256"
   },
   {
    "internalType": "string",
    "name": "_name",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "_description",
    "type": "string"
   },
   {
    "internalType": "uint256",
    "name": "_quantity",
    "type": "uint256"
   }
  ],
  "name": "updateProduct",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
 }
]

if (!window.ethereum) {
 alert('Meta Mask Not Found')
 window.open("https://metamask.io/download/")
}

export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0x4BAE03dde4D85E8E44F5Bf354C28b5f91735B3B7"

export const contract = new ethers.Contract(address, abi, signer)
```

Home.js

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
  const [Id, setId] = useState("");
  const [Name, setName] = useState("");
  const [Desc, setDesc] = useState("");
  const [Qty, setQty] = useState("");

  const [Ids, setIds] = useState("");
  const [Names, setNames] = useState("");
  const [Descs, setDescs] = useState("");
  const [Qtys, setQtys] = useState("");
  const [Wallet, setWallet] = useState("");



  const [gId, setGIds] = useState("");
  const [Details, setDetails] = useState("");
  const handleId = (e) => {
    setId(e.target.value)
  }

  const handleName = (e) => {
    setName(e.target.value)
  }

  const handleDesc = (e) => {
    setDesc(e.target.value)
  }

  const handleQty = (e) => {
    setQty(e.target.value)
  }




  const handleAddProduct = async () => {
    try {
      let tx = await contract.addProduct(Id.toString(), Name, Desc, Qty)
      let wait = await tx.wait()
      alert(wait.transactionHash)
      console.log(wait);
    } catch (error) {
      alert(error)
    }
  }


  const handleIds = (e) => {
    setIds(e.target.value)
  }

  const handleNames = (e) => {
    setNames(e.target.value)
  }

  const handleDescs = (e) => {
    setDescs(e.target.value)
  }

  const handleQtys = (e) => {
    setQtys(e.target.value)
  }



  const handleUpdate = async () => {
    try {
      let tx = await contract.updateProduct(Ids.toString(), Names, Descs, Qtys)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }


  const handleGetIds = async (e) => {
    setGIds(e.target.value)
  }

  const handleGetDetails = async () => {
    try {
      let tx = await contract.getProductDetails(gId.toString())

      let arr = []
      tx.map(e => {
        arr.push(e)
      })

      console.log(tx);
      setDetails(arr)
    } catch (error) {
      alert(error)
      console.log(error);
    }
  }

  const handleWallet = async () => {
    if (!window.ethereum) {
      return alert('please install metamask');
    }

    const addr = await window.ethereum.request({
      method: 'eth_requestAccounts',
    });

    setWallet(addr[0])

  }

 return (
  <div>
   <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Agriculture Registry</h1>
     {!Wallet ?
```

```
        <Button onClick={handleWallet} style={{ marginTop: "30px", marginBottom: "50px" }}>Connect Wallet </Button>
        :
        <p style={{ width: "250px", height: "50px", margin: "auto", marginBottom: "50px", border: '2px solid #2096f3' }}>{Wallet.slice(0, 6)}....{Wallet.slice(-6)}</p>
      }
   <Container>
    <Row>
     <Col style={{marginRight:"100px"}}>
      <div>
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleId} type="number" placeholder="Product Id" value={Id} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleName} type="string" placeholder="Product Name" value={Name} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleDesc} type="string" placeholder="product description" value={Desc} /> <br />
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleQty} type="number" placeholder="product quantity" value={Qty} /> <br />

      <Button onClick={handleAddProduct} style={{ marginTop: "10px" }} variant="primary"> Add Product</Button>
      </div>
     </Col>

            <Col style={{ marginRight: "100px" }}>
              <div>
                <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleIds} type="number" placeholder="Product Id" value={Ids} /> <br />
                <input style={{ marginTop: "10px", borderRadius: "5px" }}
    onChange={handleNames} type="string" placeholder="Product Name"
    value={Names} /> <br />
                <input style={{ marginTop: "10px", borderRadius: "5px" }}
    onChange={handleDescs} type="string" placeholder="product description"
    value={Descs} /> <br />
                <input style={{ marginTop: "10px", borderRadius: "5px" }}
    onChange={handleQtys} type="number" placeholder="product quantity"
    value={Qtys} /> <br />

                <Button onClick={handleUpdate} style={{ marginTop: "10px" }}
    variant="primary"> Update Product</Button>
              </div>
            </Col>

   </Row>
   <Row>
            <Col >
              <div style={{ margin: "auto" , marginTop:"100px"}}>
                <input style={{ marginTop: "10px", borderRadius: "5px" }}
    onChange={handleGetIds} type="number" placeholder="Enter  Id" value={gId} /><br
    />

                <Button onClick={handleGetDetails} style={{ marginTop: "10px" }}
    variant="primary">Get Product Details</Button>
                {Details ? Details?.map(e => {
                  return <p>{e.toString()}</p>
                }) : <p></p>}
              </div>
            </Col>
   </Row>
   </Container>

  </div>
     )
    }

    export default Home;
```

App.js

```
import './App.css';
import Home from './Page/Home'

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Home />
      </header>
    </div>
  );
}

export default App;
```

App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
```

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

Github link:

https://github.com/Parama02/Agriculture-Docs-Chain

Demo_video_link:

https://drive.google.com/file/d/1bQBffzDXHbTxqCYO1hu1Pm7_H-9H0tR_/view?usp=sharing