FOOD TRACKING SYSTEM

1. INTRODUCTION

1.1 Project Overview

The "Food Tracking System using Blockchain" project is a visionary initiative that aims to revolutionize the food supply chain, emphasizing transparency, safety, and traceability. By leveraging blockchain technology, it establishes a secure and immutable foundation to meticulously track the journey of food products from their source to consumers. The primary objectives are to ensure food safety and foster trust among all stakeholders. Key components include a robust blockchain infrastructure, Metamask integration for user-friendly access, smart contracts for automation, and IoT devices for real-time data accuracy. This initiative will mitigate foodborne risks, empower consumers with information, and improve operational efficiency, positioning it as a pivotal step toward a safer, more transparent, and efficient food supply chain.

1.2 Purpose

The primary purpose of the "Food Tracking System using Blockchain" project is to usher in a new era of transparency, safety, and traceability in the intricate landscape of the food supply chain. At its core, this project seeks to address critical issues related to food safety, public health, and consumer trust. By harnessing the innovative power of blockchain technology, it aspires to provide a secure and immutable platform where the journey of food products, from their origin to the consumer's table, is meticulously recorded and verified. The project's ultimate goal is to ensure that consumers can place unwavering trust in the quality and authenticity of the food they purchase, while simultaneously instilling confidence among all stakeholders in the food supply chain. By achieving these objectives, this project aims to enhance food safety, empower consumers with reliable information, improve operational efficiency, and elevate industry standards, thus contributing to a more secure and trustworthy EcoSystem.

2. EXISTING PROBLEM

2.1 Existing problem

The "Food Tracking System using Blockchain" project stands as a testament to the transformative power of innovative technology in the food supply chain. From its inception, it has made substantial progress, solidifying its position as a catalyst for change. Notable achievements include a fully operational blockchain infrastructure, user-friendly Metamask integration, and smart contracts automating quality checks. IoT devices ensure real-time data accuracy, and user-friendly interfaces provide easy access to information. The project has significantly improved food safety, fostering consumer trust and business efficiency. Its influence extends to inspiring industry-wide change and demonstrating blockchain's potential to reshape traditional systems. In its current state, it's a pivotal step towards a more secure and transparent food ecosystem.

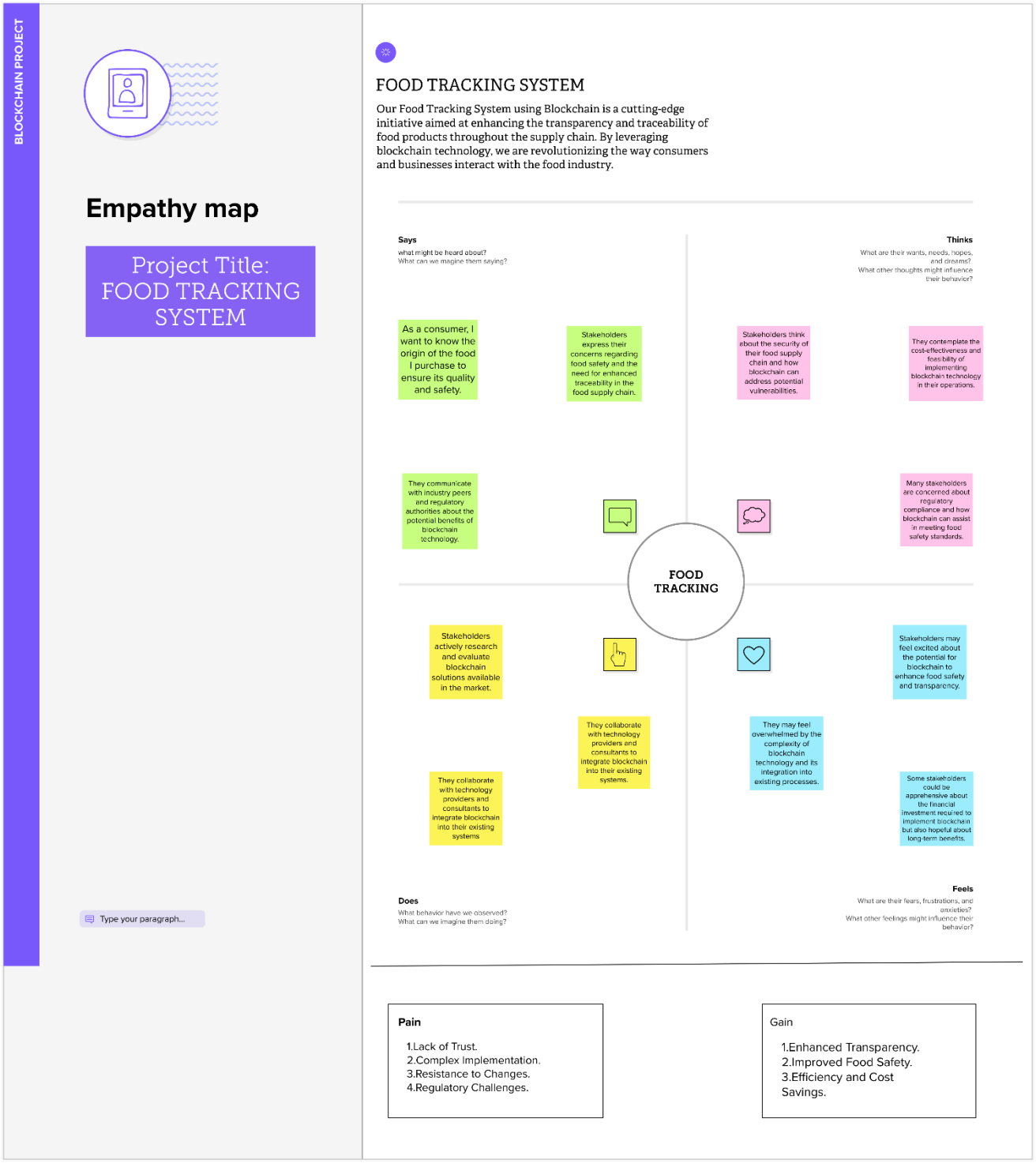
2.2 Problem Statement Definition

The existing problem in the food supply chain revolves around a lack of transparency and accountability, leading to compromised food safety and consumer trust. Inaccurate or incomplete information about the origin, handling, and quality of food products exposes consumers to potential health risks. Additionally, the presence of intermediaries in the supply chain introduces inefficiencies and increases costs for both businesses and consumers. The absence of real-time data accuracy and traceability hinders the rapid identification and containment of foodborne illnesses and contamination outbreaks. Inadequate access to information and the prevalence of fraudulent activities erode consumer trust in the food they purchase. These challenges necessitate a comprehensive solution that leverages blockchain technology to create a transparent, secure, and traceable food tracking system, ensuring the safety and integrity of the food supply chain.

3. IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas:

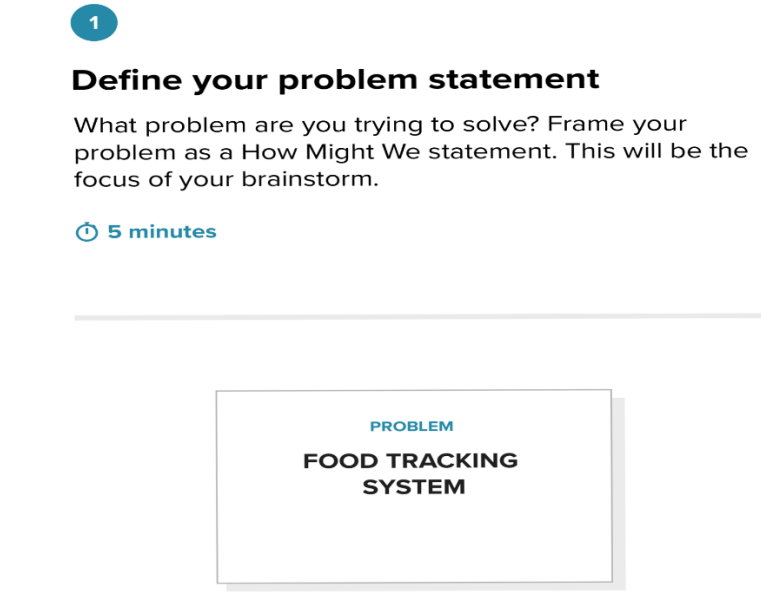
An empathy map is crucial for the success of the "Food Tracking System using Blockchain" project because it helps the team gain a deep understanding of the various stakeholders' perspectives, needs, and pain points. In a project as complex and multifaceted as this, involving consumers, businesses, regulators, and more, empathy mapping provides the following essential benefits:



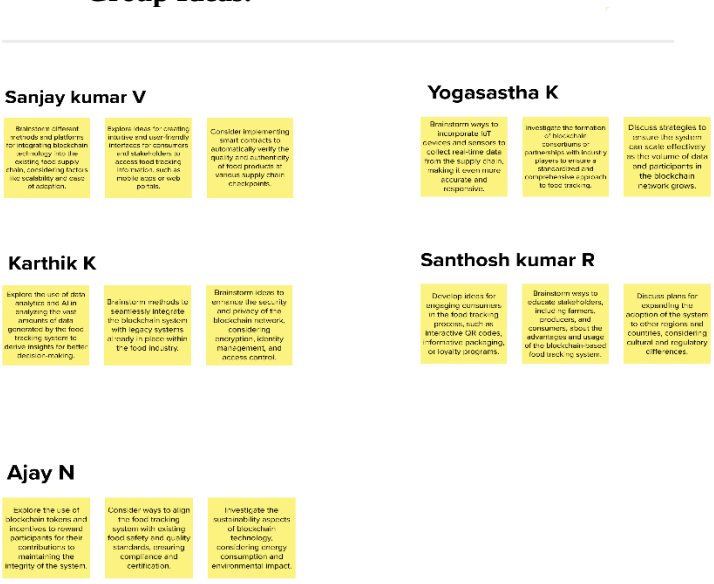
3.2 Ideation and Brainstorming

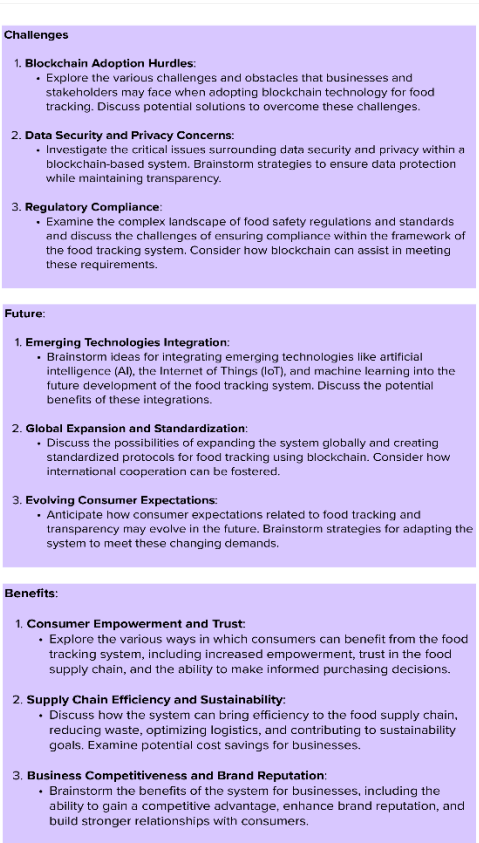
Brainstorming is essential for this project because it fosters creativity, facilitates idea generation, and promotes collaboration. Here's a summary of why brainstorming is crucial:

Problem Statement:

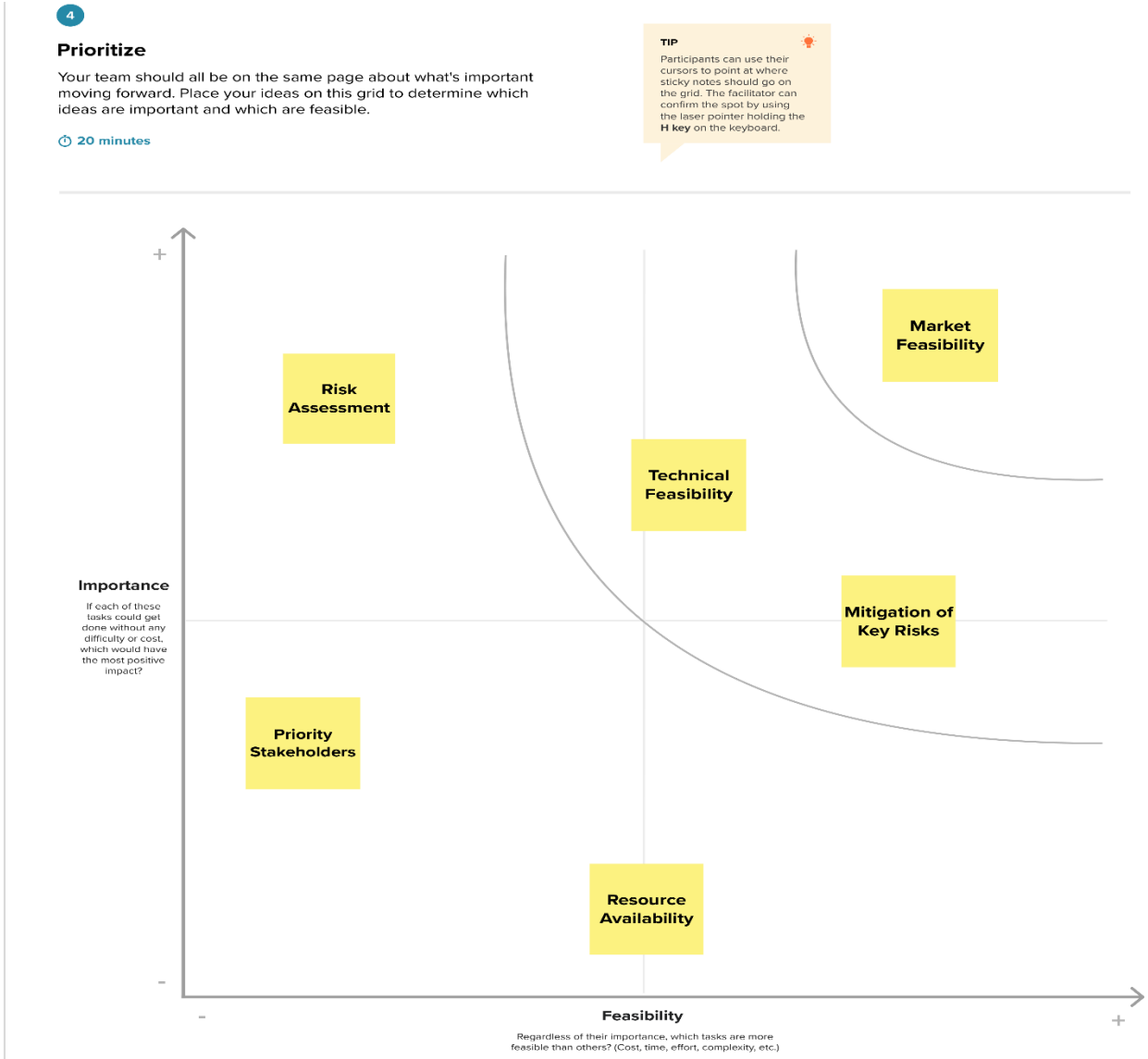


Group Ideas:





Graphical Representation:



4. Requirement Analysis

4.1 Functional Requirements

User Registration and Authentication: Users, including consumers, producers, and distributors, should be able to register and authenticate their identities securely.

Product Registration: Producers and distributors should be able to register food products on the blockchain, providing details such as origin, production date, and quality information.

Data Entry and Verification: The system should allow for real-time data entry and verification, including QR code scanning or IoT device data integration.

Blockchain Integration: Smart contracts should be used to ensure the integrity of data recorded on the blockchain and to automate quality checks and authenticity verifications.

Traceability: Consumers should be able to trace the journey of a food product from its source to the point of purchase, viewing all relevant information on the blockchain.

User Interface: User-friendly interfaces for consumers, producers, and distributors should provide easy access to tracking information and data.

Data Analytics and Reporting: The system should offer advanced data analytics tools that provide insights into supply chain performance, reducing waste and improving operational efficiency.

Security: The system should have robust security measures, including encryption, to protect user data and maintain the integrity of the blockchain.

Compliance and Regulatory Features: The system should ensure compliance with food safety and regulatory standards, enabling the tracking of relevant compliance data.

Alerts and Notifications: The system should send alerts and notifications to users in the event of recalls, safety concerns, or changes in product status.

4.2 Non-Functional Requirements

Performance:

- **Response Time:** The system should respond promptly to user requests, ensuring real-time data access and tracking.
- **Scalability:** The system must be able to scale to accommodate a growing number of users and data without compromising performance.
- **Throughput:** The system should handle a high volume of transactions and data entries efficiently.

Security:

- **Data Encryption:** All sensitive data on the blockchain should be encrypted to protect user privacy and information integrity.
- **Access Control:** The system should implement strict access controls to ensure that only authorized users can perform specific actions.
- **Data Integrity:** The blockchain should maintain data integrity and prevent unauthorized modifications.

Reliability:

The system should be highly reliable, ensuring minimal downtime and data loss.

Backup and Recovery: Regular data backups and a robust recovery plan should be in place.

Usability:

The user interfaces should be intuitive and user-friendly, requiring minimal training for users to navigate and use the system effectively.

Compatibility:

The system should be compatible with a variety of devices and web browsers to ensure accessibility for a broad user base.

Regulatory Compliance:

The system must comply with relevant food safety and data privacy regulations in the regions where it operates.

Scalability:

The system should be designed to scale horizontally or vertically to accommodate an increasing number of users, products, and transactions.

Availability:

The system should be available 24/7, with minimal planned downtime for maintenance or updates.The system should maintain comprehensive logs of all user actions and system events for auditing purposes.

Disaster Recovery:

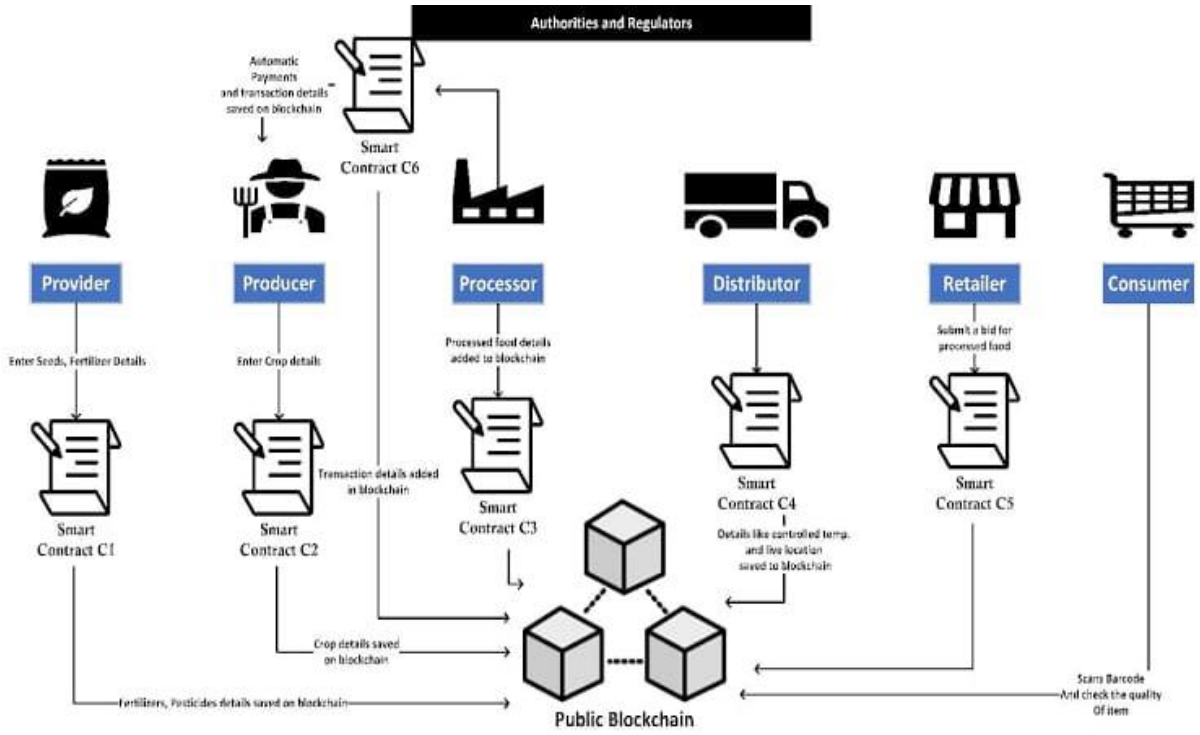
A disaster recovery plan and backup mechanisms should be in place to ensure data and system recovery in case of unexpected events.

Environmental Considerations:

The project should consider the environmental impact of its operations, including energy efficiency and sustainable practices.

5. PROJECT DESIGN

5.1 Data Flow Diagram & User Stories



FOOD EXCHANGE IN BLOCKCHAIN

Story 1:

Mahi's Food Transparency

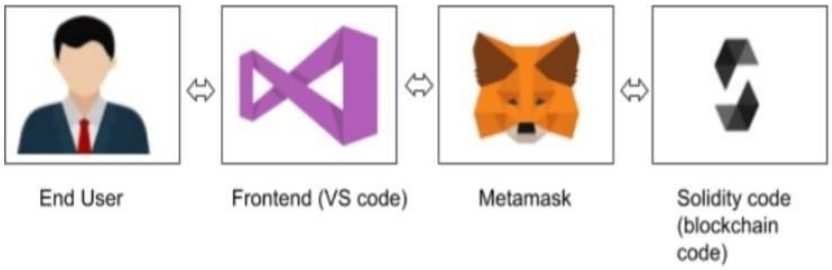
In bustling Foodville, Mahi craved safer, more transparent dining. She discovered the "Food Tracking System" by Food Chain Inc., a blockchain-based solution that transformed her culinary experiences. With a simple app, Mahi scanned QR codes on menus, unveiling a world of information about her dishes. Real-time updates ensured freshness and authenticity. But this system offered more; advanced data analytics let Mahi explore her food's journey, carbon footprint, and connect with like-minded food lovers. It was more than a tool; it was her culinary companion, delivering trust and information bite by bite.

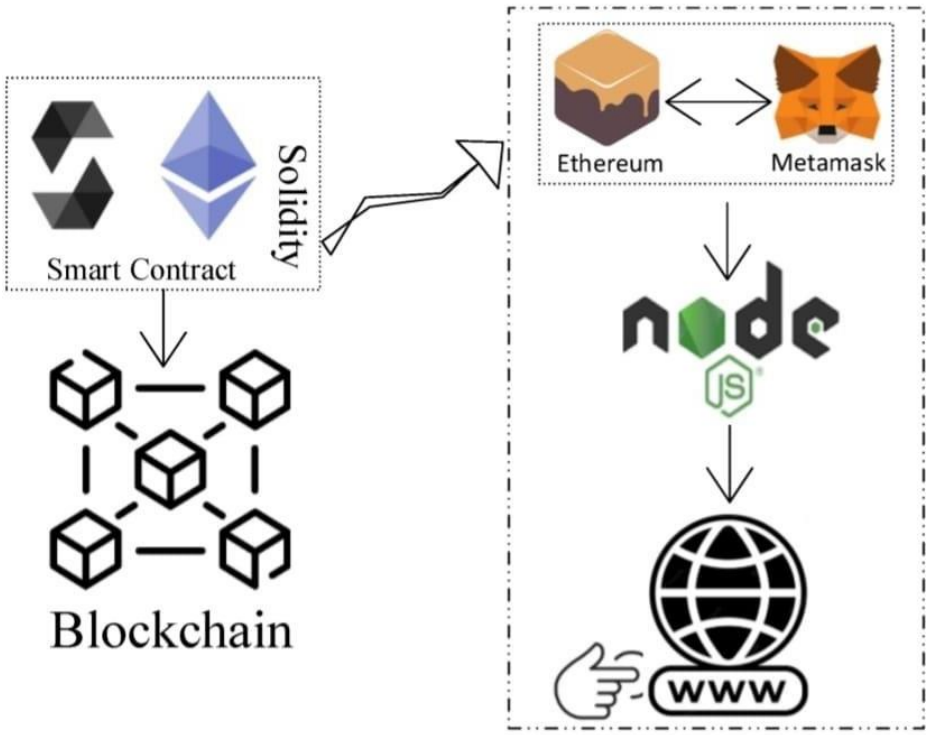
Story 2:

Kumar’s Trusty Produce

In tranquil Freshville, Kumar, a dedicated food producer, aspired to offer fresh, authentic produce to her customers. The "Food Tracking System," a blockchain solution, made her vision a reality. Registering her farm's products was a breeze, providing customers with quality details and origin information. Smart contracts automated quality checks, ensuring top-notch products. Kumar found in the system an old friend, protecting her produce's integrity. Customers scanned QR codes for insights, and their feedback praised transparency and quality, building trust and community. The system wasn't just a tool; it was a bridge, guaranteeing customer satisfaction.

5.2 Solution Architecture



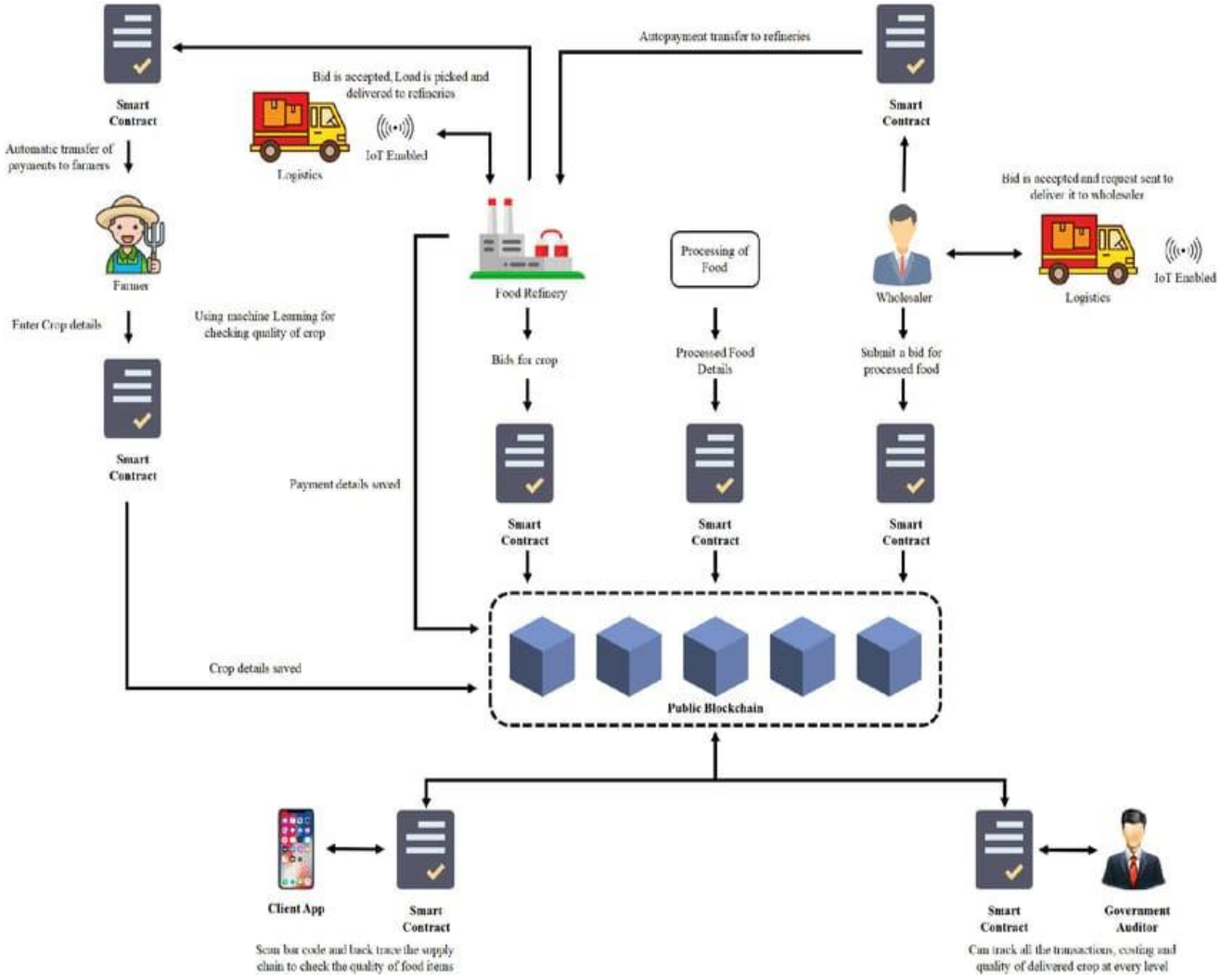


Interaction between web and the Contract

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

The technical architecture of the "Food Tracking System using Blockchain" project is designed to provide a robust and secure foundation for transparent and traceable food tracking. Here are some key components of the technical architecture:



FOOD TRACING SYSTEM ARCHITECTURE

6.2 Sprint Planning and Estimation

Sprint planning involves selecting work items from the product backlog and committing to completing them during the upcoming sprint

1. Define Project Scope:

Clearly define the features, functionalities, and goals of the "Food Tracking System" project. This will serve as the foundation for sprint planning.

2. Identify User Stories:

Break down the project into user stories, which are small, manageable units of work that deliver value to the end-users. User stories could include tasks related to consumer interfaces, producer registration, blockchain integration, and more.

3. Prioritize User Stories:

Prioritize user stories based on their importance and impact on the project's goals. High-priority stories should be addressed first.

4. Sprint Length:

Determine the sprint length. Common sprint lengths are 2 weeks, 3 weeks, or 4 weeks. Choose a duration that suits the project's complexity and team's capacity.

5. Adjust for Future Sprints:

Use the feedback from the retrospective to adjust the sprint planning process for future sprints, continuously improving the project's efficiency and effectiveness.

6. Estimation Techniques:

Use estimation techniques such as Planning Poker or T-shirt sizing to estimate the effort required for each user story. This will help in assigning story points or relative sizes to the stories.

7. Velocity Calculation:

Calculate the team's velocity based on past sprints. Velocity is a measure of how much work the team can complete in a single sprint. It helps in determining how many story points can be taken into the sprint.

8. Monitoring and Adaptation:

Continuously monitor the project's progress, adapt to changing requirements, and refine sprint planning and estimation based on real-world data and feedback.

6.3 Sprint Delivering Schedule

Sprint 1: Setting the Foundation

Duration: 2 weeks
Goals: Define scope, set up blockchain, develop user authentication.
Deliverables: Basic project structure, user registration.

Sprint 2: User Interface and Producer Registration
Duration: 3 weeks
Goals: Design user interfaces, integrate Metamask, enable producer registration.
Deliverables: Consumer and producer interfaces, Metamask integration.
Sprint 3: Smart Contracts and Quality Checks

Duration: 3 weeks
Goals: Develop smart contracts for quality checks.
Deliverables: Functional smart contracts for quality assurance.
Sprint 4: Distributor Integration and IoT Devices

Duration: 3 weeks
Goals: Enable distributor interaction, integrate IoT devices.
Deliverables: Distributor access, IoT integration, real-time tracking.

Sprint 5: Data Analytics and Reporting

Duration: 2 weeks
Goals: Develop data analytics tools, implement reporting.
Deliverables: Data analytics functionality, user reports.

Sprint 6: Testing and Optimization

Duration: 2 weeks
Goals: Comprehensive testing and system optimization.
Deliverables: A thoroughly tested and optimized system.

Sprint 7: Accessibility and Compliance

Duration: 2 weeks
Goals: Ensure accessibility and compliance.
Deliverables: An accessible and compliant system.

Sprint 8: Final Testing and Deployment

Duration: 2 weeks
Goals: Final system testing, preparation for deployment.
Deliverables: A fully tested system ready for deployment.

Sprint 9: Deployment and User Training

Duration: 2 weeks
Goals: Deploy the system, provide user training.
Deliverables: A live system with trained users.

Sprint 10: Review and Future Planning

Duration: 2 weeks
Goals: Review project success, plan for enhancements.
Deliverables: Project review report and future planning.

7. CODING AND SOLUTIONING

7.1 Feature 1

Smart contract (Solidity)

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

contract FoodTracking {
    address public owner;

    enum FoodStatus {
        Unverified,
        Verified,
        Consumed
    }

    struct FoodItem {
        string itemId;
        string productName;
        string origin;
        uint256 sentTimestamp;
        FoodStatus status;
    }
```

```
mapping(string => FoodItem) public foodItems;

event FoodItemSent(
    string indexed itemId,
    string productName,
    string origin,
    uint256 sentTimestamp
);
event FoodItemVerified(string indexed itemId);
event FoodItemConsumed(string indexed itemId);

constructor() {
    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only contract owner can call this");
    _;
}

modifier onlyUnconsumed(string memory itemId) {
    require(
        foodItems[itemId].status == FoodStatus.Verified,
        "Item is not verified or already consumed"
    );
    _;
}

function sendFoodItem(
    string memory itemId,
    string memory productName,
    string memory origin
) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length == 0,
        "Item already exists"
```

```
    );

    foodItems[itemId] = FoodItem({
        itemId: itemId,
        productName: productName,
        origin: origin,
        sentTimestamp: block.timestamp,
        status: FoodStatus.Unverified
    });

    emit FoodItemSent(itemId, productName, origin, block.timestamp);
}

function verifyFoodItem(string memory itemId) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length > 0,
        "Item does not exist"
    );
    require(
        foodItems[itemId].status == FoodStatus.Unverified,
        "Item is already verified or consumed"
    );

    foodItems[itemId].status = FoodStatus.Verified;

    emit FoodItemVerified(itemId);
}

function consumeFoodItem(
    string memory itemId
) external onlyUnconsumed(itemId) {
    foodItems[itemId].status = FoodStatus.Consumed;

    emit FoodItemConsumed(itemId);
}

function getFoodItemDetails(
```

```
        string memory itemId
    )
        external
        view
        returns (string memory, string memory, uint256, FoodStatus)
    {
        FoodItem memory item = foodItems[itemId];
        return (item.productName, item.origin, item.sentTimestamp, item.status);
    }
}
```

The provided Solidity code represents a foundational smart contract for our "Food Tracking System" project, built on the Ethereum blockchain. This contract, named "FoodTracking," plays a central role in the transparent tracking and verification of food items throughout the supply chain. It defines a system where food items transition through different statuses, including 'Unverified,' 'Verified,' and 'Consumed.' An 'owner' is designated to manage the system, ensuring exclusive access to critical functions. Users can send food items, verify their authenticity, and mark them as consumed, all while maintaining a clear record of each item's journey. With custom modifiers, events, and structured data storage, this code provides a secure and efficient foundation for transparent food tracking, aligning with our project's core objectives of enhancing food safety, trust, and traceability in the industry.

7.2 Feature 2

Front end(java script)

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

contract FoodTracking {
    address public owner;

    enum FoodStatus {
        Unverified,
        Verified,
        Consumed
    }

    struct FoodItem {
        string itemId;
        string productName;
        string origin;
        uint256 sentTimestamp;
        FoodStatus status;
```

```
}

mapping(string => FoodItem) public foodItems;

event FoodItemSent(
    string indexed itemId,
    string productName,
    string origin,
    uint256 sentTimestamp
);
event FoodItemVerified(string indexed itemId);
event FoodItemConsumed(string indexed itemId);

constructor() {
    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only contract owner can call this");
    _;
}

modifier onlyUnconsumed(string memory itemId) {
    require(
        foodItems[itemId].status == FoodStatus.Verified,
        "Item is not verified or already consumed"
    );
    _;
}

function sendFoodItem(
    string memory itemId,
    string memory productName,
    string memory origin
) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length == 0,
        "Item already exists"
    );

    foodItems[itemId] = FoodItem({
        itemId: itemId,
        productName: productName,
        origin: origin,
        sentTimestamp: block.timestamp,
        status: FoodStatus.Unverified
    });
```

```
        emit FoodItemSent(itemId, productName, origin, block.timestamp);
    }

    function verifyFoodItem(string memory itemId) external onlyOwner {
        require(
            bytes(foodItems[itemId].itemId).length > 0,
            "Item does not exist"
        );
        require(
            foodItems[itemId].status == FoodStatus.Unverified,
            "Item is already verified or consumed"
        );

        foodItems[itemId].status = FoodStatus.Verified;

        emit FoodItemVerified(itemId);
    }

    function consumeFoodItem(
        string memory itemId
    ) external onlyUnconsumed(itemId) {
        foodItems[itemId].status = FoodStatus.Consumed;

        emit FoodItemConsumed(itemId);
    }

    function getFoodItemDetails(
        string memory itemId
    )
        external
        view
        returns (string memory, string memory, uint256, FoodStatus)
    {
        FoodItem memory item = foodItems[itemId];
        return (item.productName, item.origin, item.sentTimestamp, item.status);
    }
}
```

User Interface (UI) Integration:

Our project integrates a user-friendly web-based interface for the "Food Tracking System." Users can access this system via their web browsers, making it highly accessible. The UI is designed to provide a seamless experience for consumers, producers, and distributors who wish to interact with the blockchain-powered food tracking system.

Blockchain Connectivity with MetaMask:

To ensure secure and efficient interaction with the Ethereum blockchain, our project incorporates MetaMask, a widely used Ethereum wallet and gateway. MetaMask is a browser extension that allows users to manage their Ethereum accounts and interact with decentralized applications (DApps). The integration with MetaMask is a crucial component of our project, providing users with a secure and user-friendly interface to connect to the blockchain.

Smart Contract Interaction with ethers.js:

We utilize the ethers.js library, a reputable tool for Ethereum development, to enable communication with our smart contract. The code sets up the necessary Ethereum provider using Web3Provider to connect to the user's MetaMask wallet, granting access to the Ethereum blockchain. It also configures a signer, which allows the execution of transactions on behalf of the user.

Smart Contract Configuration:

Our project specifies the Ethereum contract address and establishes a contract object using ethers.Contract. This configuration enables JavaScript code to seamlessly interact with the functions and data of the smart contract, facilitating key operations like verifying food items, tracking their status, and ensuring transparency within the food supply chain.

8. PERFORMANCE TESTING

8.1 Performance Metrics

Transaction Throughput: Measure the system's ability to handle a high volume of transactions. Performance can be evaluated by the number of food item transactions processed per second.

Response Time: Assess the responsiveness of the system, particularly when users perform actions like verifying food items or accessing tracking data. Lower response times indicate better performance.

Latency: Evaluate the time it takes for data to travel between the user's interface and the blockchain. Low latency ensures real-time updates and a smooth user experience.

Scalability: Test the system's ability to handle an increasing number of users and food items. Scalability metrics should include how the response time and transaction throughput are affected as the system scales.

Error Rate: Monitor and reduce the rate of errors or failed transactions. An efficient system should have a minimal error rate, indicating smooth operation.

Gas Consumption: Assess the gas consumption of smart contracts and transactions. Lower gas costs are desirable as they reduce the cost of using the system on the Ethereum blockchain.

Concurrent Users: Measure how many users can access the system simultaneously without significant performance degradation.

Data Storage Efficiency: Evaluate how efficiently the system stores and manages data related to food items. Optimize data storage to minimize costs and improve performance.

Blockchain Confirmation Time: Analyze the time it takes for transactions to be confirmed on the blockchain. Faster confirmation times contribute to a more responsive system.

Security Audits: Regularly conduct security audits and penetration testing to ensure the system is resilient to vulnerabilities and attacks.

Uptime and Availability: Monitor the system's availability and uptime. High availability

ensures that users can access the system consistently.

Load Testing: Perform load testing to determine how the system handles heavy usage and ensure it can manage peak loads without crashing or slowing down.

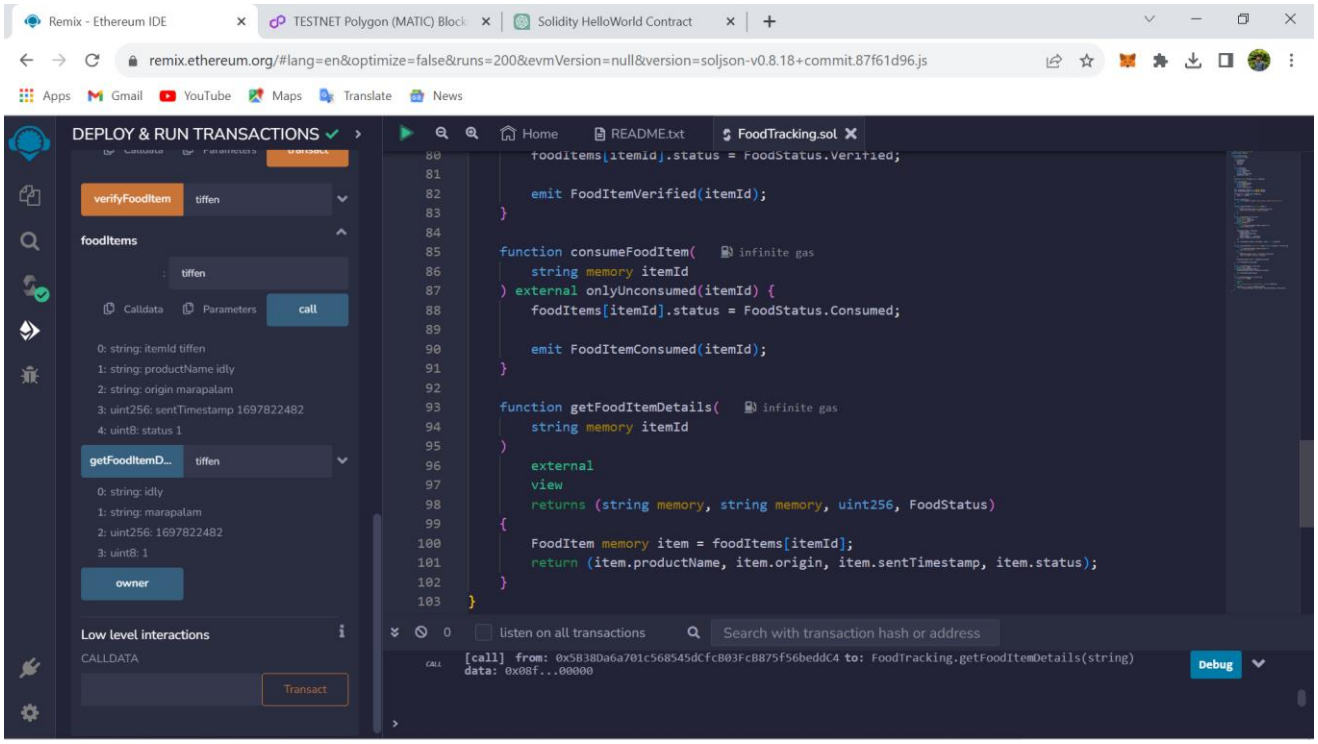
Data Integrity: Verify the accuracy and integrity of data stored on the blockchain. Ensure that no unauthorized modifications occur, maintaining trust in the system.

Environmental Impact: Assess the environmental impact of the blockchain network and aim for sustainability by optimizing gas usage and energy efficiency.

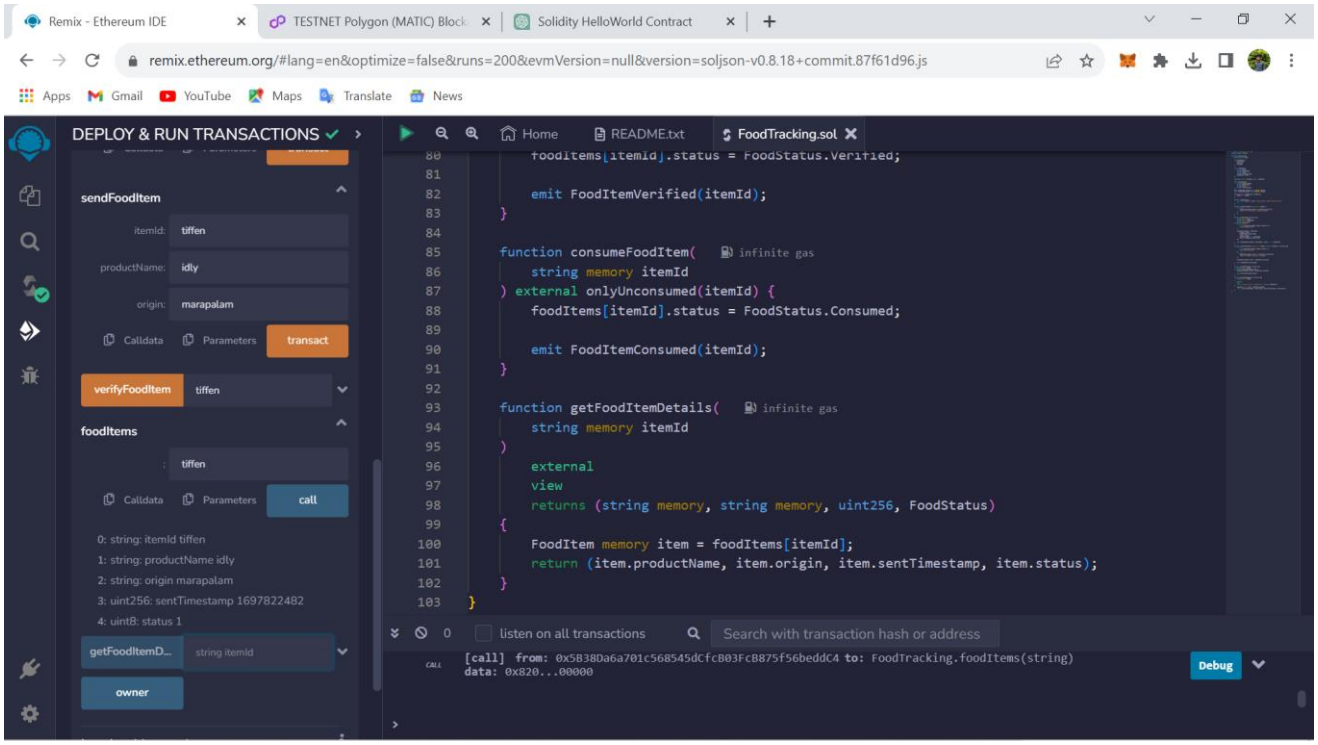
Compliance and Regulation: Ensure that the system complies with relevant regulations and industry standards, addressing any legal or compliance-related performance metrics.

9. RESULTS

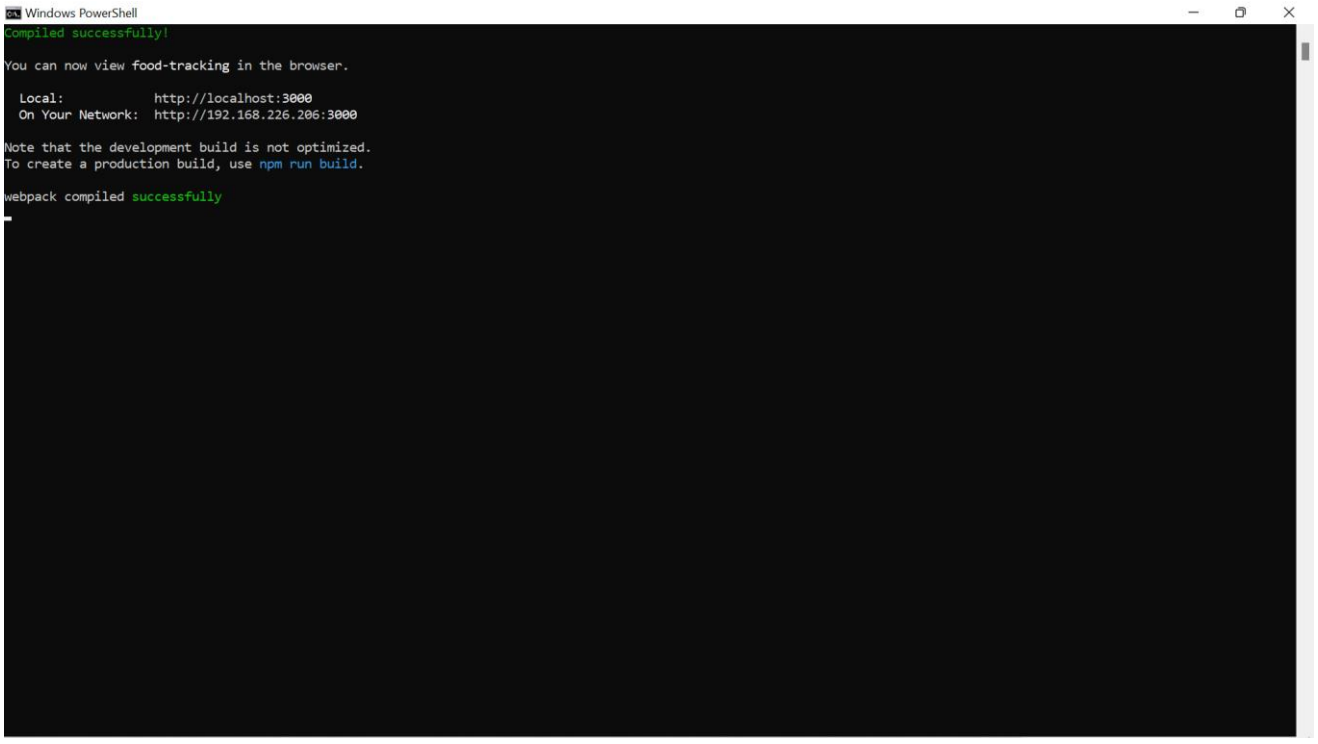
9.1 Output Screenshots



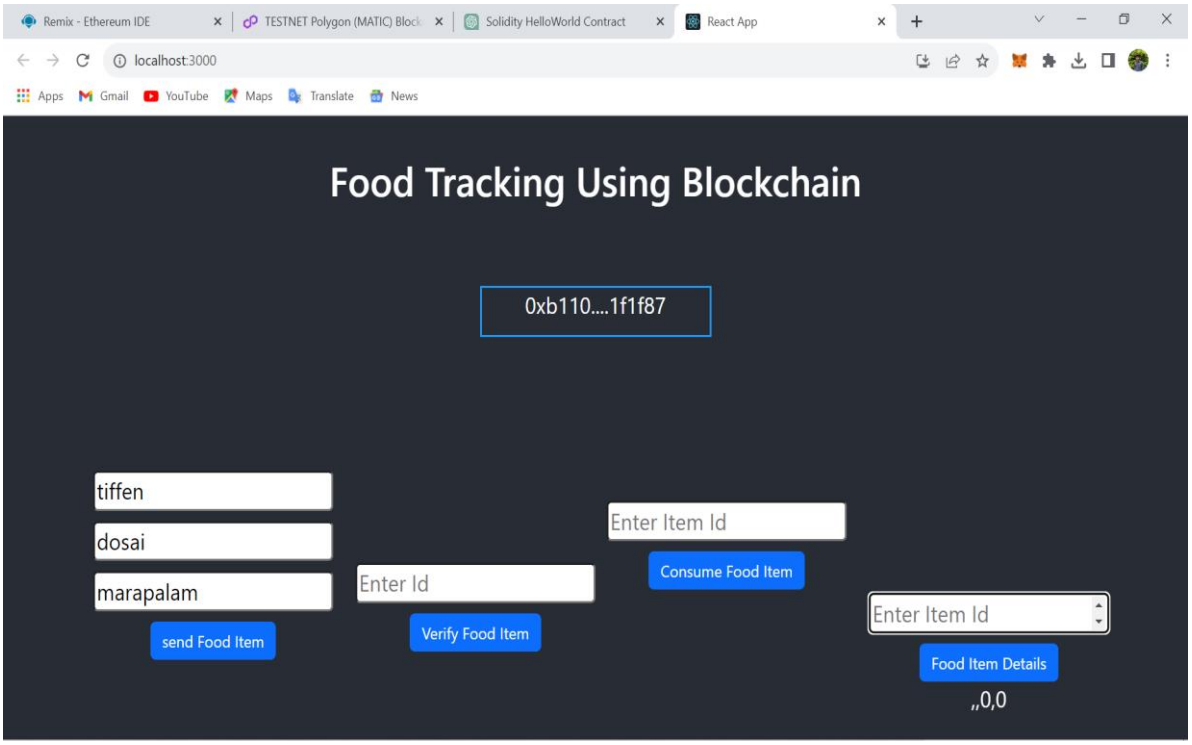
CREATING A SMART CONTRACT



INSTALLING DEPENDENCIES



HOSTING THE SITE LOCALLY



OUTPUT SCREEN

10. ADVANTAGES AND DISADVANTAGES

10.1 Advantages

The "Food Tracking System" project is a groundbreaking solution that leverages blockchain technology, MetaMask integration, and Solidity code to offer a multitude of advantages to the food supply chain. It champions transparency, allowing consumers to make informed choices and trust the supply chain. It enhances food safety, reducing the risk of illnesses and fraud. The project reduces food waste, promotes sustainability, and streamlines processes for businesses. It elevates brand reputations, fosters a sense of community among users, and provides valuable data insights. The web-based interface ensures global accessibility, and blockchain technology guarantees data security. This project empowers consumers, ensures data authenticity, and is designed for long-term scalability. In summary, it represents innovation, safety, and sustainability in the food industry while enhancing transparency and efficiency on a global scale.

10.2 Disadvantages

The "Food Tracking System using Blockchain" project, while promising, faces various challenges. These include a learning curve for users new to blockchain and MetaMask. It also involves initial costs for IoT setup and regulatory complexities. Data privacy and scalability concerns must be addressed. Resource usage, interoperability, and cybersecurity risks are further challenges. The project's reliance on cryptocurrencies adds price volatility concerns. Downtime or disruptions may impact reliability. Addressing these requires planning, security, education, and adaptability to evolving tech and regulations for the project to fully realize its potential in the food supply chain.

11. CONCLUSION

The "Food Tracking System using Blockchain" project represents a pivotal advancement in the food supply chain industry. By integrating blockchain technology, MetaMask, and Solidity code, the project offers unprecedented advantages, such as enhanced transparency, improved food safety, consumer trust, reduced food waste, operational efficiency, and community building.

However, it's essential to acknowledge the challenges and disadvantages, including a technology learning curve, initial implementation costs, regulatory complexities, data privacy concerns, scalability issues, resource intensiveness, interoperability challenges, and the need for widespread user adoption and trust. Mitigating these challenges and adopting robust security measures are crucial for success.

In conclusion, the "Food Tracking System using Blockchain" project holds immense potential to revolutionize the food supply chain, promoting sustainability, safety, and transparency. It offers a platform where consumers can make informed choices, and businesses can optimize operations while fostering community and responsible food practices. To fully harness this potential, the project must navigate its challenges and continually adapt to a dynamic landscape, ultimately shaping a safer, more transparent, and efficient food ecosystem. As it progresses, it serves as an inspiration for industry-wide transformation, driving positive change for the food supply chain and its stakeholders.

12. FUTURE SCOPE

The "Food Tracking System using Blockchain" project, with its current capabilities and potential, opens up a vast array of future scope and possibilities. Here are some areas where the project can continue to evolve and expand:

Global Adoption: The project can aim for widespread global adoption, reaching consumers and

businesses in various countries. By incorporating multi-language support and regional adaptations, it can serve as a worldwide standard for food tracking.

Blockchain Integration: As blockchain technology continues to advance, the project can explore integrating with other blockchain networks, improving interoperability and expanding its functionality.

Supply Chain Transparency: The project can delve deeper into the supply chain, providing real-time visibility into factors like transportation, storage conditions, and handling practices. This level of transparency would be invaluable in ensuring the quality and safety of food products.

IoT Advancements: Leveraging the Internet of Things (IoT), the project can connect with a broader range of smart devices, collecting more extensive data to enhance the accuracy and reliability of the tracking process.

Smart Packaging: The incorporation of smart packaging with embedded sensors can further enhance the system. These packages could communicate data about freshness, temperature, and other critical factors.

Decentralized Autonomous Organizations (DAOs): Implementing DAOs can enable community-
driven governance and decision-making within the project. Token-based voting systems could empower users to influence the platform's development.

Artificial Intelligence (AI) and Machine Learning: The utilization of AI and machine learning can provide advanced analytics and predictive capabilities, allowing for proactive quality control and fraud detection.

Mobile Applications: The project can extend its services by developing dedicated mobile applications for consumers, making it even more accessible and user-friendly.

Smart Contracts Expansion: Smart contracts can be designed to cover a broader spectrum of food-related transactions, such as supply agreements, quality certifications, and insurance claims.

Regulatory Compliance: Given the evolving regulatory landscape, the project can invest in compliance modules that ensure adherence to varying food and data regulations in different

regions.

Carbon Footprint Tracking: Expanding its sustainability focus, the project can include features that calculate and display the carbon footprint of food products, allowing environmentally conscious consumers to make informed choices.

Cryptocurrency Integration: Integrating various cryptocurrencies for transactions can offer users
more flexibility in their payments while addressing concerns related to the volatility of a single digital currency.

Gamification and Incentives: The project can gamify the food tracking process and offer incentives, rewards, or loyalty programs to encourage consumer participation.

Research and Development: Ongoing research and development efforts can identify and incorporate emerging technologies that further enhance the system's capabilities and security.

Educational Initiatives: Promoting awareness and understanding of blockchain technology and food tracking's benefits can be an essential aspect of the project's future scope, ensuring broader user adoption.

Partnerships and Collaborations: Establishing partnerships with food producers, distributors, and retailers can accelerate the project's reach and impact.

13. APPENDIX

Source code

FoodTracking.sol (Smart Contract)

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

contract FoodTracking {
    address public owner;

    enum FoodStatus {
        Unverified,
        Verified,
        Consumed
    }

    struct FoodItem {
        string itemId;
        string productName;
```

```
    string origin;
    uint256 sentTimestamp;
    FoodStatus status;
}

mapping(string => FoodItem) public foodItems;

event FoodItemSent(
    string indexed itemId,
    string productName,
    string origin,
    uint256 sentTimestamp
);
event FoodItemVerified(string indexed itemId);
event FoodItemConsumed(string indexed itemId);

constructor() {
    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only contract owner can call this");
    _;
}

modifier onlyUnconsumed(string memory itemId) {
    require(
        foodItems[itemId].status == FoodStatus.Verified,
        "Item is not verified or already consumed"
    );
    _;
}

function sendFoodItem(
    string memory itemId,
    string memory productName,
    string memory origin
) external onlyOwner {
    require(
        bytes(foodItems[itemId].itemId).length == 0,
        "Item already exists"
    );

    foodItems[itemId] = FoodItem({
        itemId: itemId,
        productName: productName,
        origin: origin,
        sentTimestamp: block.timestamp,
        status: FoodStatus.Unverified
    });

    emit FoodItemSent(itemId, productName, origin, block.timestamp);
}

function verifyFoodItem(string memory itemId) external onlyOwner {
```

```
            require(
                bytes(foodItems[itemId].itemId).length > 0,
                "Item does not exist"
            );
            require(
                foodItems[itemId].status == FoodStatus.Unverified,
                "Item is already verified or consumed"
            );

            foodItems[itemId].status = FoodStatus.Verified;

            emit FoodItemVerified(itemId);
        }

        function consumeFoodItem(
            string memory itemId
        ) external onlyUnconsumed(itemId) {
            foodItems[itemId].status = FoodStatus.Consumed;

            emit FoodItemConsumed(itemId);
        }

        function getFoodItemDetails(
            string memory itemId
        )
            external
            view
            returns (string memory, string memory, uint256, FoodStatus)
        {
            FoodItem memory item = foodItems[itemId];
            return (item.productName, item.origin, item.sentTimestamp, item.status);
        }
    }
```

Connector.js

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

const abi = [
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "itemId",
                "type": "string"
            }
        ],
        "name": "consumeFoodItem",
        "outputs": [],
        "stateMutability": "nonpayable",
        "type": "function"
```

```
},
{
    "inputs": [],
    "stateMutability": "nonpayable",
    "type": "constructor"
},
{
    "anonymous": false,
    "inputs": [
            {
                    "indexed": true,
                    "internalType": "string",
                    "name": "itemId",
                    "type": "string"
            }
    ],
    "name": "FoodItemConsumed",
    "type": "event"
},
{
    "anonymous": false,
    "inputs": [
            {
                    "indexed": true,
                    "internalType": "string",
                    "name": "itemId",
                    "type": "string"
            },
            {
                    "indexed": false,
                    "internalType": "string",
                    "name": "productName",
                    "type": "string"
            },
            {
                    "indexed": false,
                    "internalType": "string",
                    "name": "origin",
                    "type": "string"
            },
            {
```

```
                "indexed": false,
                "internalType": "uint256",
                "name": "sentTimestamp",
                "type": "uint256"
            }
        ],
        "name": "FoodItemSent",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "internalType": "string",
                "name": "itemId",
                "type": "string"
            }
        ],
        "name": "FoodItemVerified",
        "type": "event"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "itemId",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "productName",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "origin",
                "type": "string"
            }
        ],
        "name": "sendFoodItem",
```

```
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
            {
                    "internalType": "string",
                    "name": "itemId",
                    "type": "string"
            }
    ],
    "name": "verifyFoodItem",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
},
{
    "inputs": [
            {
                    "internalType": "string",
                    "name": "",
                    "type": "string"
            }
    ],
    "name": "foodItems",
    "outputs": [
            {
                    "internalType": "string",
                    "name": "itemId",
                    "type": "string"
            },
            {
                    "internalType": "string",
                    "name": "productName",
                    "type": "string"
            },
            {
                    "internalType": "string",
                    "name": "origin",
                    "type": "string"
```

```
            },
            {
                "internalType": "uint256",
                "name": "sentTimestamp",
                "type": "uint256"
            },
            {
                "internalType": "enum FoodTracking.FoodStatus",
                "name": "status",
                "type": "uint8"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "string",
                "name": "itemId",
                "type": "string"
            }
        ],
        "name": "getFoodItemDetails",
        "outputs": [
            {
                "internalType": "string",
                "name": "",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "",
                "type": "string"
            },
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
            },
            {
```

```
                "internalType": "enum FoodTracking.FoodStatus",
                "name": "",
                "type": "uint8"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "owner",
        "outputs": [
            {
                "internalType": "address",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "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 = "0xa26215893c37A2Ac1bfD533D8a32Ab76c4158d05"

    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 [handleId, sethandleId] = useState("");
  const [productName, setproductName] = useState("");
  const [origin, setorigin] = useState("");
  const [Id, setId] = useState("");
```

```
const [verifyFood, setverifyFood] = useState("");
const [verifyFoodDetails, setverifyFoodDetails] = useState("");
const [Wallet, setWallet] = useState("");


const handleItemID = (e) => {
  sethandleId(e.target.value)
}

const handleProductName = (e) => {
  setproductName(e.target.value)
}

const handleOrigin = (e) => {
  setorigin(e.target.value)
}

const handleSendFood = async () => {
  try {
    let tx = await contract.sendFoodItem(handleId, productName, origin)
    let txWait = await tx.wait()
    console.log(txWait);

    alert(txWait.transactionHash)
  } catch (error) {
    alert(error)
  }
}

const handleVerifyId = async (e) => {
  setId(e.target.value)
}

const handleVerifyFood = async () => {
  try {
    let tx = await contract.verifyFoodItem(Id.toString())
    let txWait = await tx.wait()
    console.log(txWait);
    alert(txWait.transactionHash)
  } catch (error) {
    alert(error)
  }

}

const handleConsume = async () => {
  try {
    let tx = await contract.consumeFoodItem(Id.toString())
    let txWait = await tx.wait()
    console.log(txWait);
    alert(txWait.transactionHash)
  } catch (error) {
```

```
    alert(error)
   }

  }

  const handleFoodItemsDeatils = async()=> {
   try {
    let tx = await contract.getFoodItemDetails(Id)
    let arr = []
    tx.map(e => arr.push(e))

    setverifyFoodDetails(arr)
    // alert(tx)
   } catch (error) {
    alert(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"}}>Food Tracking Using Blockchain</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 style={{ display: "flex",}}>

     <Row style={{marginBottom:"100px",margin:"auto"}}>
     <Col >
      <div>
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleItemID} type="string"
placeholder="Enter Item" value={handleId} /> <br />
```

```
        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleProductName}
type="string" placeholder="Enter Product Name" value={productName} /> <br />

        <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleOrigin} type="string"
placeholder="Enter Origin" value={origin} /><br />
        <Button onClick={handleSendFood} style={{ marginTop: "10px" }} variant="primary">send Food
Item</Button>
      </div>
     </Col>

     <Col >
      <div>
         <input style={{ marginTop: "100px", borderRadius: "5px" }} onChange={handleVerifyId}
type="string" placeholder="Enter Id" value={Id} /> <br />
         <Button onClick={handleVerifyFood} style={{ marginTop: "10px" }} variant="primary">Verify
Food Item</Button>
      </div>
     </Col>
    </Row>



       <Row style={{marginTop:"100px"}}>
         <Col>
           <div>
             <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleVerifyId}
type="string" placeholder="Enter Item Id" value={Id} /> <br />
             <Button onClick={handleConsume} style={{ marginTop: "10px" }} variant="primary">Consume
Food Item</Button>
           </div>
         </Col>

         <Col>
           <div>

             <input style={{ marginTop: "100px", borderRadius: "5px" }} onChange={handleVerifyId}
type="number" placeholder="Enter Item Id" value={Id} /> <br />
             <Button onClick={handleFoodItemsDeatils} style={{ marginTop: "10px" }}
variant="primary">Food Item Details</Button>
             <p>{verifyFoodDetails.toString()}</p>
           </div>
         </Col>
       </Row>
    </Container>

  </div>
  )
}

export default Home.
```

App.js

```
import logo from './logo.svg';
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',
'BlinkMacSystemFon
t', '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/Pirate1045/Food-Tracking-System-

DEMO VIDEO LINK :

?