INGREDIENTS TRACING USING BLOCKCHAIN FOR RAPID PRODUCT RECALL

BY

CHIN YI HAO

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr. Aun Yichiet, for providing me with the invaluable opportunity to work on a blockchain development project. This experience has marked an important first step toward building my career in the field of blockchain technology, and I am deeply thankful for his guidance and support.

I am also profoundly grateful to my parents and family for their unwavering love, support, and encouragement throughout the course of this project. Their constant motivation has been a source of strength, without which the successful completion of this work would not have been possible.

ABSTRACT

Food safety and traceability remain critical challenges in modern supply chains, where contamination, mislabeling, unsafe storage conditions, and delayed recall procedures can result in serious public health risks and economic losses. Traditional traceability systems frequently depend on centralized databases, manual records, and paper-based documentation, making them vulnerable to tampering, incomplete reporting, and slow investigation during food safety incidents. This project presents NutriGuard, a blockchain-based food traceability and recall system designed to improve transparency, accountability, and response speed across three main stages of a food supply chain: ingredient registration, product production, and consumer verification. NutriGuard combines Ethereum-compatible smart contracts, a Flutter mobile application, a Raspberry Pi edge gateway, and IoT sensors. Supplier, ingredient, product, quality-control, consumer-registration, and feedback records are stored through smart contract transactions, while selected off-chain references such as certificate or product document hashes are represented by IPFS hash fields. During production, merchants define HACCP-inspired quality rules for each product. Temperature and humidity can be retrieved automatically from the Raspberry Pi edge gateway through a Flask REST API, while weight and pH remain manually entered in the current prototype. Smart contracts validate submitted production data against the product's quality thresholds and only allow QR code generation when the production record is compliant. For consumers, NutriGuard provides QR-based product verification, product registration for recall alerts, traceability information, and feedback submission. For merchants, the system provides supplier management, ingredient management, product creation, quality control, recall initiation, affected-product discovery, and feedback processing. When an ingredient is marked as contaminated or recalled, the smart contract automatically updates all affected products to a contaminated state and blocks QR code usage, supporting rapid recall actions.

Area of Study (Minimum 1 and Maximum 2): Blockchain

Keywords (Minimum 5 and Maximum 10): Blockchain, IoT, Ethereum, Smart Contracts, DApp, Product Recall, Food Security

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

T	Actual temperature reading in Celsius
T_min	Minimum acceptable temperature threshold
T_max	Maximum acceptable temperature threshold
H	Actual relative humidity percentage
H_min	Minimum acceptable humidity threshold
H_max	Maximum acceptable humidity threshold
W	Actual product or ingredient weight in grams
pH	Acidity/alkalinity value (stored as integer ×100 in smart contract)
n	Number of valid sensor samples in the sliding window

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
API	Application Programming Interface
B2B	Business-to-Business
CAPA	Corrective and Preventive Action
CID	Content Identifier
DApp	Decentralized Application
EVM	Ethereum Virtual Machine
FDA	Food and Drug Administration
GPIO	General Purpose Input/Output
HACCP	Hazard Analysis and Critical Control Point
HUS	Hemolytic Uremic Syndrome
IoT	Internet of Things
IPFS	InterPlanetary File System
JSON	JavaScript Object Notation
MUS	Melamine-related Urolithiasis
NTP	Network Time Protocol
OOS	Out of Specification
OOT	Out of Trend
PoS	Proof of Stake
QR	Quick Response
REST	Representational State Transfer
RPC	Remote Procedure Call
RPi	Raspberry Pi
SDK	Software Development Kit
SMTP	Simple Mail Transfer Protocol
TSPR	Traceability System for Product Recall
UI	User Interface
UPC	Universal Product Code

Chapter 1
Introduction

Food supply chains are increasingly distributed across farms, suppliers, processors, restaurants, logistics providers, retailers, and consumers. As this network grows, the ability to verify food origin, monitor safety conditions, and perform rapid recalls becomes more difficult. Centralized record systems can store useful data, but they also introduce trust concerns because records can be modified by privileged administrators, lost during system migration, or delayed by manual reporting. These limitations become especially serious when contamination has already reached consumers and investigators must identify affected batches quickly.

Blockchain and IoT technologies provide a suitable foundation for a more transparent traceability system. Blockchain offers an immutable, append-only transaction log that can preserve supplier, ingredient, product, quality-control, recall, and consumer-registration events. Smart contracts can enforce business rules automatically, such as preventing QR code generation when production conditions fail quality checks. IoT devices can complement blockchain by collecting real-world environmental data such as temperature and humidity, reducing dependence on manual entry.

This project, NutriGuard, implements a prototype food traceability and recall system using a Solidity smart contract, a Flutter mobile application, and a Raspberry Pi edge gateway connected to IoT sensors. It is designed for a simulated restaurant environment in which merchants register suppliers, ingredients, and products, submit production quality data, generate QR codes for safe products, and initiate recalls when contamination is detected. Consumers scan QR codes to view traceability information, register products for recall alerts, and submit feedback.

1.1 Problem Statement and Motivation

Traditional food traceability systems typically depend on centralized databases, manual data entry, or paper-based documentation, all of which are vulnerable to human error, fraud, and delayed response times. When a contaminated or unsafe product is detected,

identifying its origin and distribution path can be time-consuming, leading to increased health risks, financial losses, and reputational damage.

Numerous real-world food contamination incidents highlight the limitations of existing systems. According to the case in [1], China suffered from the melamine-tainted milk powder scandal in 2008, in which melamine was illegally added to infant formula to raise protein test values falsely. The scandal affected at least 294,000 children, with approximately 52,000 hospitalized for MUS and six deaths. The scandal exposed loopholes in food safety supervision and attracted widespread attention. Another Example is the romaine lettuce contamination incident in the United States in 2018. Based on the investigation summary report in [2], From October to December 2018, 62 cases of E. coli O157:H7 infection were reported in 16 United States and Canada, including 25 hospitalizations and 2 cases of HUS. Epidemiological surveys showed that 83% of patients had eaten romaine lettuce a week before the onset of the disease. After that, the FDA tracked the supply chain and locked in a farm in the Santa Maria area of Santa Barbara County, California. The E. coli O157:H7 strain that matched the epidemic was detected in the sediment samples of the farm's reservoir.

In both cases, the failure to identify the source of contamination promptly resulted in serious illness or even death. In addition, because it took weeks to track down the source of contamination, it brought significant public health risks and economic losses to manufacturers and retailers. These incidents demonstrate the urgent need for an efficient, transparent, and tamper-proof traceability system to shorten the time required to identify, isolate, and recall contaminated food. In addition, the existing system provides limited transparency to consumers, who often lack access to reliable and detailed information about the food they consume. Therefore, a decentralized, automated solution that integrates blockchain technology and IoT-based real-time monitoring is needed to overcome these challenges and enhance food safety management.

This project is driven by the growing prevalence of food safety incidents and the limitations of existing traceability systems. Blockchain, defined as a distributed digital ledger that structures data into sequentially linked “blocks” forming a “chain” [3],

offers a promising foundation for addressing these challenges. This structure improves the security of the information because the credibility of new blocks must be verified and accepted by most nodes in the network. The integration of blockchain and IoT technologies has shown potential in creating transparent, tamper-proof, and decentralized solutions across various industries. Ethereum is a decentralized, Turing-complete blockchain platform that enables programmable smart contracts and dApps through its built-in EVM [4]. It achieves secure consensus via PoS, enhancing energy efficiency while supporting complex stateful logic beyond simple asset transfers [5]. Ethereum's versatility and robust developer ecosystem make it a foundational infrastructure for diverse Web3 and enterprise use cases, from finance to supply chain.

Complementing this, IoT devices can continuously capture critical parameters such as temperature, humidity, and embedding this data immutably on-chain. By leverage blockchain's transparency, immutability, and decentralization to ensure data integrity, this system not only enhances traceability but also empowers consumers with verifiable product histories. According to [6], smart contracts are self-executing agreements coded on blockchain networks, offering automation, transparency, security, and cost-efficiency by eliminating intermediaries. Smart contracts cannot be modified once deployed, so they have zero tolerance for even minor outliers throughout the entire processing step. Therefore, a smart contract layer allows for automated quality control by enforcing predefined safety standards and triggering alerts when deviations occur. This enables faster response and targeted recall actions, minimizing health risks and financial losses.

1.2 Objectives

The main objective of this project is to design and implement NutriGuard, a blockchain-based food traceability and recall prototype that integrates mobile interaction and IoT-assisted quality control. The specific objectives are:

1. To design and develop a blockchain-based food traceability architecture that records supplier, ingredient, product, production, recall, consumer registration, and feedback data through Ethereum-compatible smart contracts.
2. To implement a smart-contract-based quality validation mechanism that enforces HACCP-inspired production thresholds and prevents QR code

generation when submitted production data fails the defined safety requirements.

3. To integrate an IoT-assisted production monitoring module using a Raspberry Pi edge gateway and environmental sensors to retrieve temperature and humidity data during the quality-control process.
4. To design and implement an ingredient-level rapid recall mechanism that automatically identifies affected products, updates their safety status, and blocks QR-based verification when a contaminated or recalled ingredient is detected.
5. To develop a mobile-based merchant and consumer interface that supports supplier and ingredient management, product creation, QR code generation, product verification, recall checking, product registration, and consumer feedback submission.

1.3 Project Scope and Direction

The project focuses on a working prototype rather than a production-scale deployment. The scope includes the following:

1. **Blockchain traceability:** Supplier, ingredient, product, quality-control, recall, consumer-registration, and feedback data are recorded through an Ethereum-compatible Solidity smart contract deployed on a local Hardhat blockchain for development and demonstration.
2. **Merchant application:** Merchants can register suppliers, register ingredients, create products, define HACCP-inspired quality rules, submit production data, generate QR codes for compliant products, initiate recalls, and process consumer feedback.
3. **Consumer application:** Consumers can scan QR codes, view product details, verify ingredient and production status, register for recall alerts, view registered products, and submit feedback.
4. **IoT edge gateway:** Raspberry Pi 3B+ with a DHT22 sensor collects temperature and humidity data. A Flask REST API allowing the Flutter application to fetch readings and auto-fill production data.
5. **Notification scope:** Consumers can register an email address on-chain for product alerts.

1.4 Contributions

This project presents both system-level and technical contributions in the development of the NutriGuard platform. The system contribution focuses on delivering an integrated food traceability and recall prototype consisting of the following components:

- A merchant application for managing suppliers, ingredients, and product records
- A consumer application for QR-based product verification and feedback submission
- A QR code traceability mechanism for accessing product lifecycle information
- A recall-aware validation system that reflects contamination status in real time
- An alert and feedback dashboard to support post-consumption monitoring and decision-making

These components collectively enhance transparency in the food supply chain and provide a structured mechanism for identifying and managing potentially unsafe products.

From a technical perspective, the system introduces several key design and implementation contributions:

- A Solidity-based smart contract system for enforcing quality validation rules
- An ingredient-level recall cascade mechanism, enabling affected product tracking through dependency mapping
- A hybrid on-chain/off-chain data architecture, where blockchain stores critical metadata and IPFS hash references are used to support data integrity verification
- An IoT gateway integration that collects environmental data for production quality assessment
- A state-driven validation model implemented within smart contracts to manage product lifecycle transitions

Unlike traditional centralized systems, this approach improves data transparency and traceability while maintaining a balance between decentralization and system practicality.

1.5 Report Organization

This report is organized into seven chapters. Chapter 1 introduces the project motivation, problem statement, objectives, scope, and contributions. Chapter 2 reviews related blockchain traceability and recall systems and positions NutriGuard against existing approaches. Chapter 3 presents the methodology, including architecture, use cases, activity diagrams, smart contract design, and IoT edge gateway design. Chapter 4 describes the system design, component specifications, interactions, and security considerations. Chapter 5 explains the implementation of hardware, software, configuration, and system operation with screenshot placeholders. Chapter 6 evaluates the system using functional testing, IoT gateway testing, blockchain transaction evaluation, and objective achievement analysis. Chapter 7 concludes the project and recommends future improvements.

Chapter 2
Literature Review

2.1 Blockchain recall management in pharmaceutical industry

A research paper [7] proposed a blockchain framework for pharmaceutical recalls to address longstanding problems in pharmaceutical recalls, such as low efficiency, poor data transparency, and high risk of tampering. This framework introduces a three-layer architecture comprising IoT-enabled data collection, a blockchain network with smart contracts, and an application layer integrating regulatory systems like OOS/OOT management and CAPA.

One of the core functions of the system is synergistic decision. Firstly, the authors designed a model to analyse the data read in real-time from the blockchain network to obtain complaint rates and adverse events, then submit the results to smart contracts for recognition. If the results meet the recall conditions and trigger the recall classification, the smart contract will instruct the relevant laboratory to conduct the OOS/OOT investigation. After that, the laboratory will conduct an automated OOS/OOT investigation according to the set formula and upload the analysis results to the network of blockchain, while combining the consensus protocol to guarantee that the investigation process cannot be tampered with. Therefore, the recall procedure becomes transparent and auditable. In addition, the framework facilitates active and passive recalls, driven by automated decisions and stakeholder opinions.

The pharmaceutical case offers some valuable insights into food traceability and recalls systems. It first emphasizes how important data standardization and intelligent gateways are to enabling blockchain compatibility. Besides, it demonstrates how smart contracts can automatically enforce recall standards, which reduce human error and speed up response times.

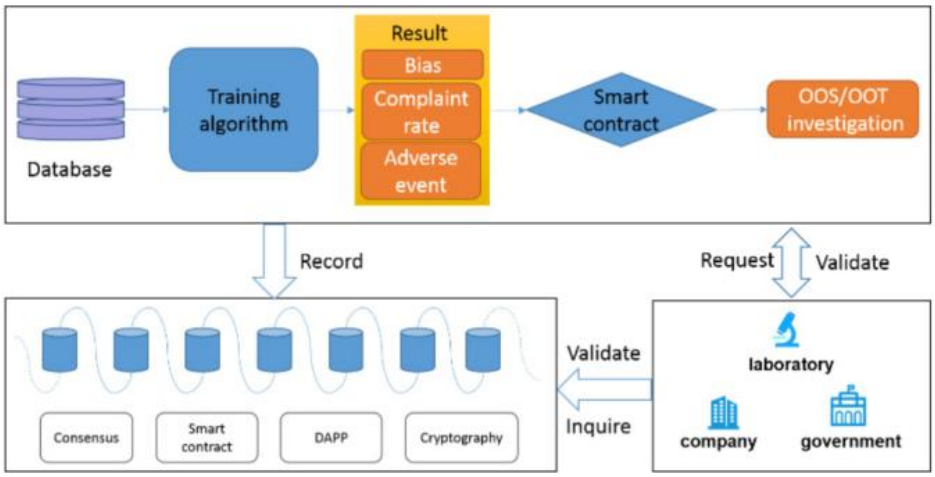


Figure 2.1 Product quality detection and recall process

2.1.1 Strengths and Weakness

This pharmaceutical recall framework showcases several notable strengths. Firstly, it presents a structured, layered architecture that ensures modularity, with clear separation between IoT data acquisition, blockchain processing, and application-level functionality. This modularity facilitates scalability and ease of integration with existing regulatory platforms such as OOS/OOT and CAPA. Secondly, the use of automated smart contracts to analyse complaint rates and adverse events significantly reduces reliance on manual intervention, thereby enhancing recall efficiency and minimizing human error. The introduction of synergistic decision-making further enables both active and passive recall procedures, ensuring a more comprehensive and timely response. The immutable audit trail created by blockchain and consensus protocols adds transparency and traceability to every investigation step, addressing core trust issues in pharmaceutical recalls.

However, several limitations also exist. The system heavily depends on predefined rules and static thresholds for triggering recalls, which may not adapt well to evolving patterns or complex scenarios that require contextual judgment. The reliance on high-quality IoT infrastructure and data standardization may also pose implementation challenges in less digitized environments. Moreover, the study does not explore consumer-facing applications or public verification mechanisms, which limits its usefulness in domains like food safety, where end-user trust and transparency are essential. Finally, the system's technical complexity and cost may hinder adoption, especially for smaller enterprises or in regions with limited blockchain expertise.

2.2 Tracing Textile and Clothing Value Chain Using Blockchain Technology

Alves et al. [8] proposed a blockchain architecture based on Hyperledger Fabric to track environmental scores and labour practices in the textile and clothing value chain to improve the transparency and sustainability of the textile and apparel industry.

The system integrates smart contracts with IoT data and off-chain storage, allowing consumers to verify the lifecycle and environmental footprint of each product lot via a DApp. The system's on-chain data model is divided into two core entities, Lot representing the batch and activity of the product, and then smart contracts are used to ensure the logical consistency of the batch and activity. For example, the batch usage must match the input batch total. Besides that, the author emphasizes the idea of a digital twin, where each physical product lot is associated with a digital representation that is updated through production, logistics, and reception activities. The author defined multiple transaction methods and embedded business rules when designing smart contracts. The first is data integrity verification, such as prohibiting the modification of completed logistics activities. The second point is ownership transfer, which automatically updates the batch owner information. Moreover, the architecture includes the use of oracle services to bridge on-chain and off-chain data, facilitating the integration of external sustainability metrics such as carbon emissions or recycled material ratios.

In conclusion, the authors propose a complete blockchain architecture to achieve transparent traceability of the textile and apparel value chain. The system also embeds business rules through the implementation of smart contracts to ensure data credibility and process compliance.

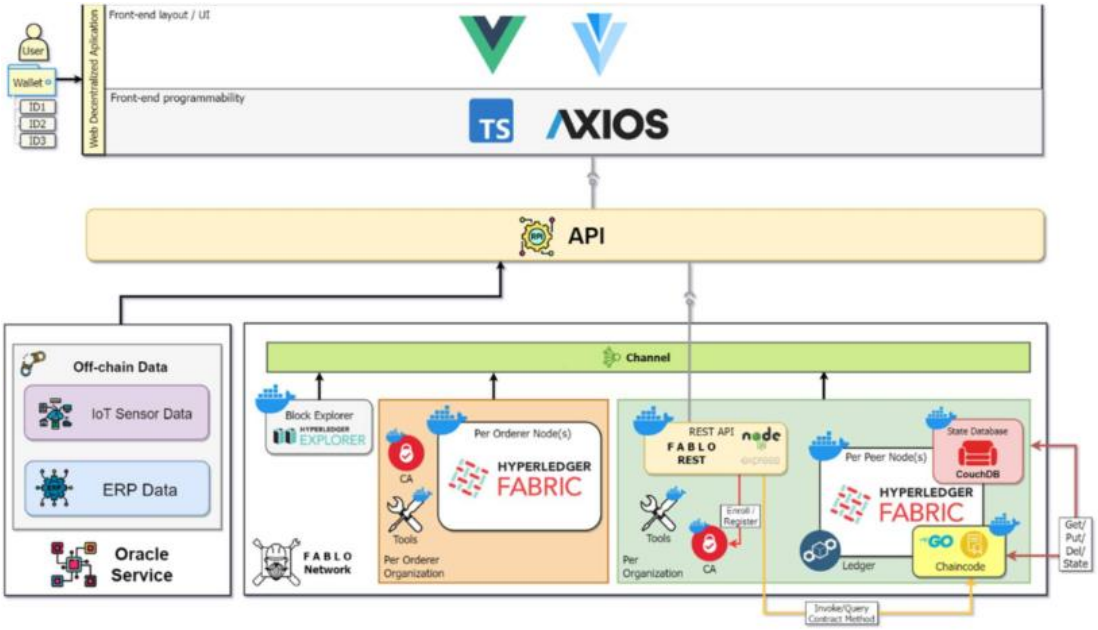


Figure 2.2 Platform architecture

2.2.1 Strengths and Weakness

The blockchain architecture proposed by Alves et al. exhibits several notable strengths. One of the key advantages is the use of digital twins, which ensures that every physical product lot is accurately mirrored by a digital entity on the blockchain. This facilitates comprehensive traceability across the production, logistics, and reception stages. The system's modular smart contract design with embedded business logic for data integrity and ownership transfer which helps automate key validation processes, reducing human error and enforcing compliance in a decentralized manner. Furthermore, the integration of oracle services allows the system to incorporate off-chain sustainability metrics, such as carbon footprint and recycled content ratios, expanding its relevance beyond transactional data and supporting Environmental, Social, and Governance objectives. The consumer-facing DApp also adds value by offering end users transparent insights into product provenance, aligning with rising demand for ethical and sustainable consumption.

Despite these strengths, the system also has several limitations. First, its reliance on Hyperledger Fabric, a permissioned blockchain, may limit openness and cross-platform interoperability, especially compared to public blockchains like Ethereum. The off-chain data dependency for sustainability metrics introduces potential trust issues unless robust verification mechanisms are in place. Additionally, while the system enables

activity-level tracking, it does not appear to support automated anomaly detection or recall mechanisms, which are crucial in safety-critical supply chains like food or pharmaceuticals.

2.3 Blockchain-based Traceability System for Product Recall

The paper [9] proposes the TSPR, a blockchain-based framework developed on Ethereum to enhance the transparency and efficiency of recall management across supply chains.

The TSPR system is designed to involve all major stakeholders including manufacturers, supply chain participants, regulators such as FDA, administrators, and end consumers within a unified blockchain network. It supports end-to-end product tracking and implements smart contracts to automate the product recall process based on predefined conditions and role authorizations. TSPR is built on five main components: role identification, product traceability, product recall for supply chain actors and regulators, and Ethereum smart contracts. These components ensure that product events such as manufacturing, distribution, and retailing are immutably logged on the blockchain, enabling rapid trace-and-track capabilities during food safety incidents. Apart from that, this framework also integrates EPCIS and RES for enhanced event tracking and regulatory compliance.

In conclusion, experimental results from their prototype demonstrate successful smart contract execution, accurate role-based access control, and full traceability of product movement through the chain. The system integrates supply chain participants, regulators, administrators, and consumers through blockchain to build a transparent and tamper-proof recall management framework.

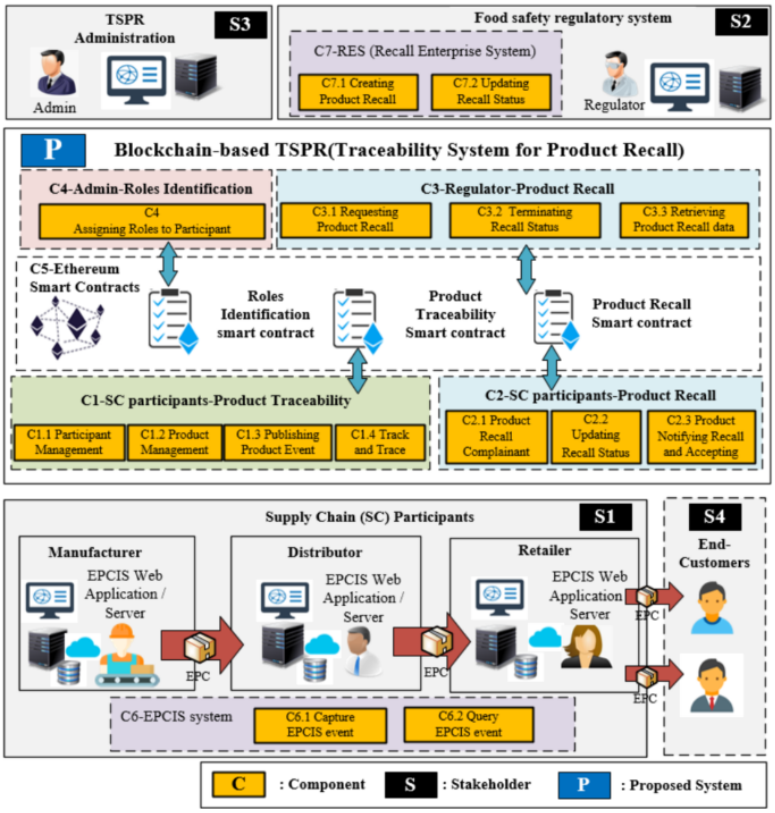


Figure 2.3 Framework of the blockchain-based TSPR system

2.3.1 Strengths and Weakness

The TSPR system presents several key strengths that make it particularly relevant to food and pharmaceutical supply chains. One of its most significant advantages is its comprehensive stakeholder inclusion, bringing together manufacturers, regulators, administrators, supply chain actors, and consumers within a single blockchain ecosystem. This holistic approach ensures that each entity plays a role in the traceability and recall process, facilitating coordinated responses and accountability. Secondly, the use of smart contracts to automate recall actions based on predefined conditions and role-based access control reduces administrative overhead and minimizes the risk of human error or delayed interventions. The integration of EPCIS and RES also reflects a commitment to industry-standard compliance, which enhances interoperability and regulatory alignment. The system's prototype demonstrates not only technical feasibility on Ethereum but also robustness in traceability and auditability.

However, the system also has some limitations. The event-driven model relies heavily on accurate and timely data input, which, if compromised, could still lead to misinformation despite the immutability of blockchain. Unlike more advanced

frameworks, TSPR does not incorporate IoT sensor integration, meaning that real-time environmental data such as temperature or humidity for perishable goods which is not captured automatically. This makes the system less responsive to quality deviations that may occur during transport or storage. Additionally, while it offers full traceability, there is no mention of dynamic product representation or a built-in incentive mechanism to encourage proactive data submission or consumer engagement. Lastly, scalability and transaction cost concerns typical to public blockchains like Ethereum are not explicitly addressed, which may affect system performance in large-scale implementations.

2.4 Blockchain-based Framework Using Smart Contracts for Supply Chain Collaboration

Research [10] present a blockchain-based framework that leverages smart contracts to support supply chain collaboration, particularly in complex and dynamic business ecosystems. Unlike traditional supply chains with fixed hierarchies and centralized information systems, the authors focus on resource-sharing networks where participants are decentralized, relationships are fluid, and trust is a major concern. Their proposed framework addresses this by enabling autonomous interactions, secure data validation, and privacy-preserving order distribution using smart contracts deployed on the Ethereum blockchain.

The research develops a multi-layered architecture involving actors such as suppliers, OEMs, and a PDU. In addition, the supply chain network is divided into multiple independent channels in the design of the network architecture, each of which connects a specific supplier and manufacturer to ensure data privacy and security. Smart contracts are used to automate order allocation based on supplier capacity and product quality, while verifying product quality through sensor data and automatically triggering payment or return processes. Besides, ensure that each channel runs smart contracts independently. For example, PDU only accesses necessary data such as production capacity and quality level to avoid information leakage. Next, the authors simulated a collaborative network consisting of two suppliers and two manufacturers for testing and demonstrated the order allocation, production quality verification, and payment processes. In the test of deploying the contract, they successfully ran the smart contract on the Ethereum test network and displayed the order allocation and

transaction records through the Etherscan interface, proving the feasibility of the framework.

In conclusion, this paper proposes an innovative blockchain framework that enables transparent

collaboration and resource optimization in supply chain networks through smart contracts, providing theoretical support and practical paths for building trust and improving efficiency in complex business ecosystems.

2.4.1 Strengths and Weakness

The framework offers several important strengths, especially for collaborative and decentralized supply chain environments. A major contribution is its support for dynamic, non-hierarchical networks, allowing suppliers and manufacturers to interact without relying on a central authority. This flexibility reflects real-world business ecosystems where partnerships can be ad hoc and rapidly changing. The use of independent blockchain channels for each supplier-manufacturer pair enhances data confidentiality by ensuring that only authorized parties can access sensitive information, addressing one of the major challenges in multi-party collaboration. Furthermore, the smart contracts are not only used for transaction logging but also for automating operational logic, such as order allocation, payment release, and quality-based returns to streamline processes and reducing manual intervention. The integration of sensor-based quality verification introduces a layer of objective, real-time validation, improving trust in product assessment. The successful test deployment on the Ethereum network and demonstration using Etherscan reinforce the technical viability of the approach.

Nonetheless, the framework has several limitations. Firstly, while the design enables data privacy through channel isolation, this increases architectural complexity, particularly when the number of suppliers and manufacturers grows, potentially leading to management overhead. Secondly, the system focuses primarily on B2B collaboration and resource sharing, without extending visibility or transparency to regulators or end consumers. This situation limiting its utility in scenarios where public traceability and trust are essential, such as food or pharmaceutical recalls. Additionally, although the

authors mention sensor-based quality data, the integration of IoT components is not elaborated in detail, leaving uncertainty about real-time monitoring capabilities.

2.5 A Prototype of Supply Chain Traceability using Solana as blockchain and IoT

M. Ashraf and C. Heavey [11] present a prototype that utilizes the Solana blockchain to achieve traceability in a multi-stage supply chain network. The system focuses on enhancing transparency and efficiency in tracking product transitions across entities such as suppliers, manufacturers, distributors, and retailers. Unlike Ethereum, the authors chose Solana for its high throughput, low transaction cost, and sub-second finality, making it suitable for supply chains that require fast and frequent data updates.

The architecture includes a web-based DApp that interacts with Solana smart contracts using the Anchor framework and the Phantom wallet for identity management. Each stakeholder is assigned a unique account, and smart contracts are designed to record and update product status at each stage of the supply chain. A unique product id is used to track the product as it moves from one actor to another, with metadata such as timestamp, actor role, and geographic location stored immutably on-chain.

To maintain simplicity, the authors implemented a minimal set of functionalities including product creation, transfer, and verification. The prototype demonstrates how each supply chain actor can append their operations on-chain in a tamper-proof manner, creating an auditable trail that consumers and regulators can verify. The web UI allows any product to be queried using its product id, ensuring full traceability across the lifecycle.

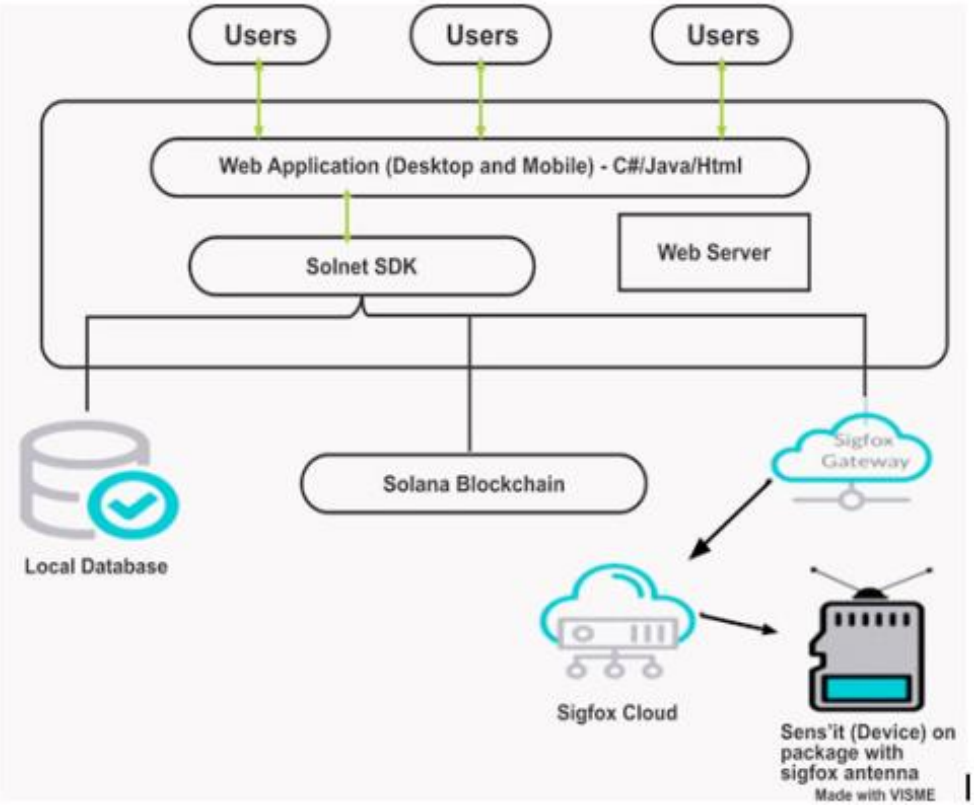


Figure 2.4 Overview of the system's structure

2.5.1 Strengths and Weakness

This Solana-based prototype brings several compelling strengths. First and foremost, its use of the Solana blockchain addresses the scalability and cost issues often associated with Ethereum. Solana's low latency and high throughput make it particularly well-suited for real-time applications in supply chains, where frequent updates are necessary. The system's simple yet functional smart contract logic allows for essential traceability actions such as product handoffs and status updates without overcomplicating the transaction flow. Additionally, the integration of the Anchor framework and Phantom wallet streamlines both contract deployment and user interaction, improving developer productivity and user experience. The approach also emphasizes immutability and role-based tracking, reinforcing transparency and accountability across the supply chain.

However, several limitations are evident. The prototype does not incorporate IoT integration, which limits its ability to collect or verify real-time environmental data, a crucial element in many safety-critical industries. The system also lacks token-based incentives, consumer interaction features, or recall automation capabilities—

components that are increasingly expected in modern traceability systems. Moreover, while Solana's performance is advantageous, it introduces platform-specific dependencies such as Anchor, a Rust-based smart contracts, which may pose a learning curve for teams accustomed to EVM-compatible ecosystems. Lastly, the framework mainly addresses B2B transparency but provides limited mechanisms for regulatory compliance or end-user engagement, reducing its scope in consumer-centric supply chains.

2.6 Summary of Existing System

The reviewed studies demonstrate the potential of blockchain technology to enhance traceability, transparency, and accountability across diverse industries including pharmaceuticals, textiles, and supply chains in general. Several common strengths can be observed. First, blockchain provides an immutable audit trail, enabling stakeholders to verify data integrity and reducing the risk of tampering. Second, the integration of smart contracts allows automation of business logic, such as recall procedures, ownership transfers, and compliance verification, thereby minimizing human error and improving operational efficiency. Third, consumer-facing applications and digital representations, such as DApps and digital twins, enable greater end-user trust by offering visibility into product provenance and sustainability metrics. Additionally, the use of IoT devices and oracle services further expands the scope of traceability systems by linking on-chain data with real-world conditions.

Despite these advantages, the existing systems also face several limitations. Many frameworks remain heavily dependent on predefined rules and static thresholds, which may lack adaptability in dynamic environments. Some solutions rely on permissioned blockchains, limiting openness and interoperability. Others fail to incorporate real-time IoT integration, making them less responsive to quality deviations during storage and transport. Furthermore, consumer engagement is often limited, with few mechanisms for direct verification, feedback, or automated recall notifications. Scalability and cost remain pressing concerns, particularly in public blockchain deployments, while technical complexity poses barriers for small enterprises. These gaps highlight the need for a system that combines blockchain's transparency with real-time IoT monitoring, automated quality enforcement, and active consumer participation.

2.7 Proposed Solution Positioning

NutriGuard is positioned as a lightweight, mobile-first, Ethereum-compatible food traceability prototype. It combines key strengths from existing research while addressing several practical gaps.

References	Blockchain	IoT Integration	Recall Automation	Consumer Interaction	Main Limitation
Pharmaceutical recall framework [7]	Private Blockchain	Yes	Yes	Limited	Complex regulatory setup
Textile traceability framework [8]	Hyperledger Fabric	Partial	No	Yes	Not food-safety focused
TSPR product recall [9]	Ethereum	No	Yes	Limited	No real-time sensor input
Smart contract collaboration [10]	Ethereum	sensor-based concept	Partial	No	B2B focus
Solana traceability prototype [11]	Solana	No	No	Limited	No recall or IoT quality control
NutriGuard	Ethereum	Yes	Ingredient-level cascade	Flutter merchant and consumer app	Local prototype, production hardening needed

Table 2.1 Comparative Analysis of Blockchain-based Traceability Frameworks

Compared with the reviewed systems, NutriGuard uniquely combines a mobile merchant workflow, QR-based consumer verification, HACCP-inspired smart contract validation, Raspberry Pi edge sensing, and ingredient-level recall cascade in one prototype.

Chapter 3

System Methodology/Approach

3.1 Methodology

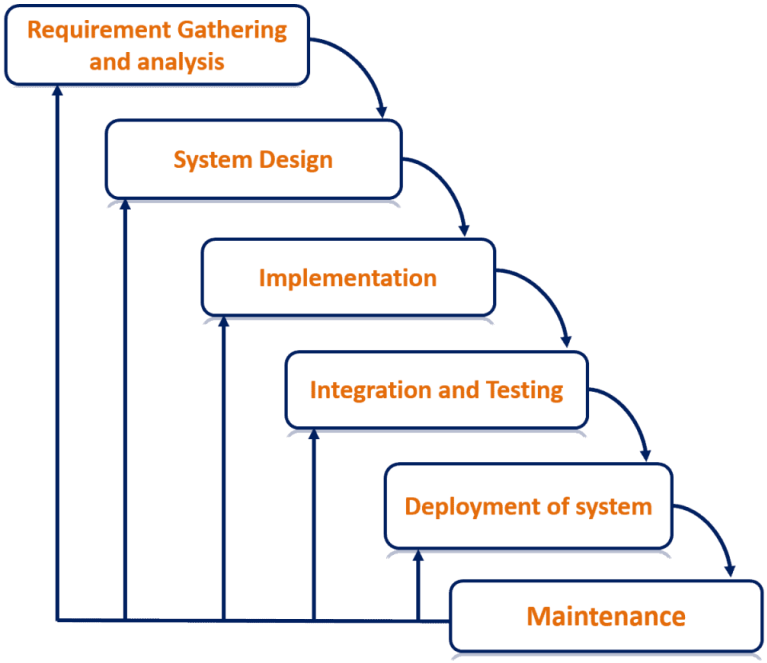


Figure 3.1 Waterfall Model

The processes of the project were categorized into different phases in the development according to the waterfall model in the SDLC, which were requirement analysis, system and software design, system implementation, system integration and unit testing, deployment of system, and maintenance.

However, due to the hybrid nature of the system involving blockchain, IoT, and mobile applications, an additional operational workflow model is introduced to better describe real system behavior. The system operates through the following structured workflow:

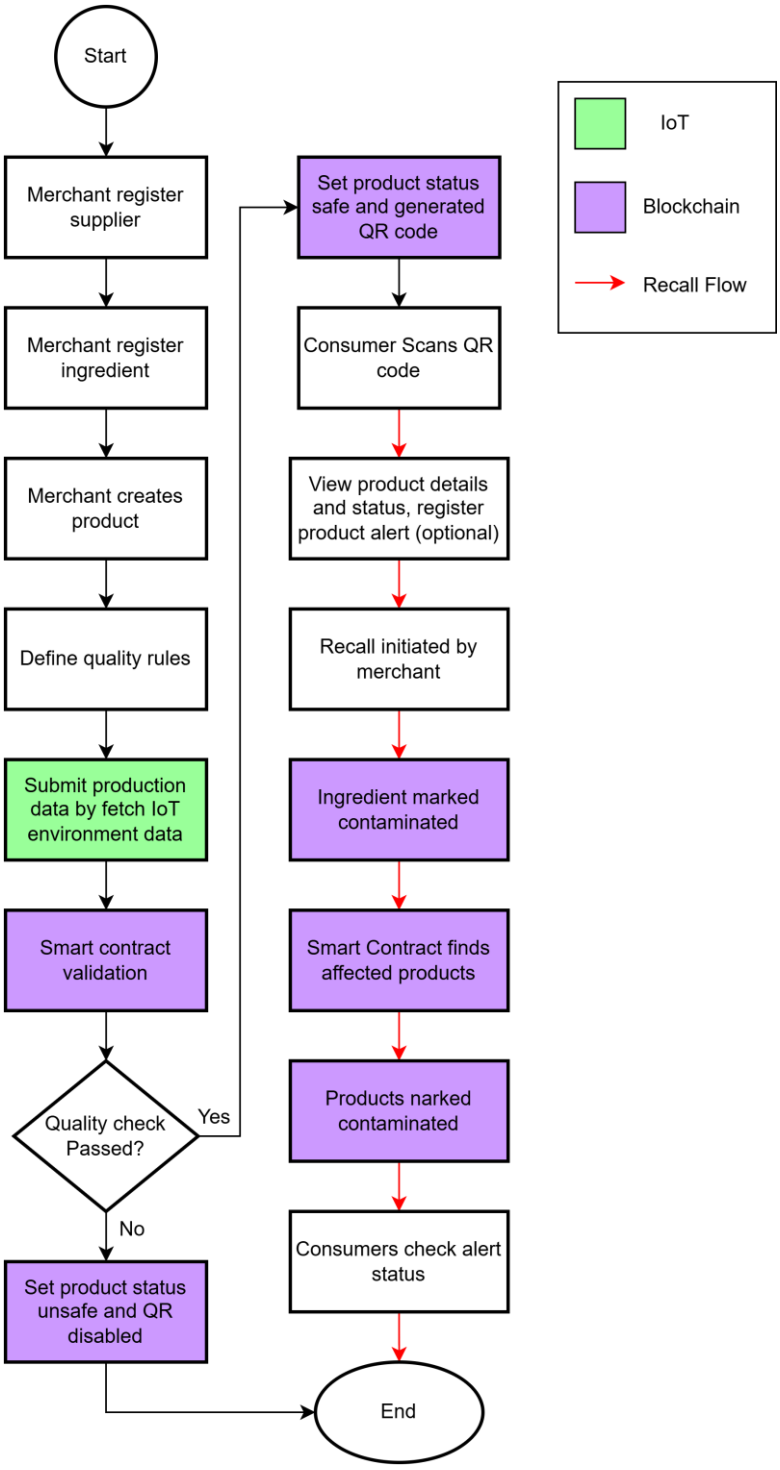


Figure 3.2 System Workflow Diagram

This workflow complements the Waterfall model by illustrating how different system components interact during runtime, particularly in handling quality validation and recall scenarios.

3.2 System Design Diagram

3.2.1 System Architecture Diagram

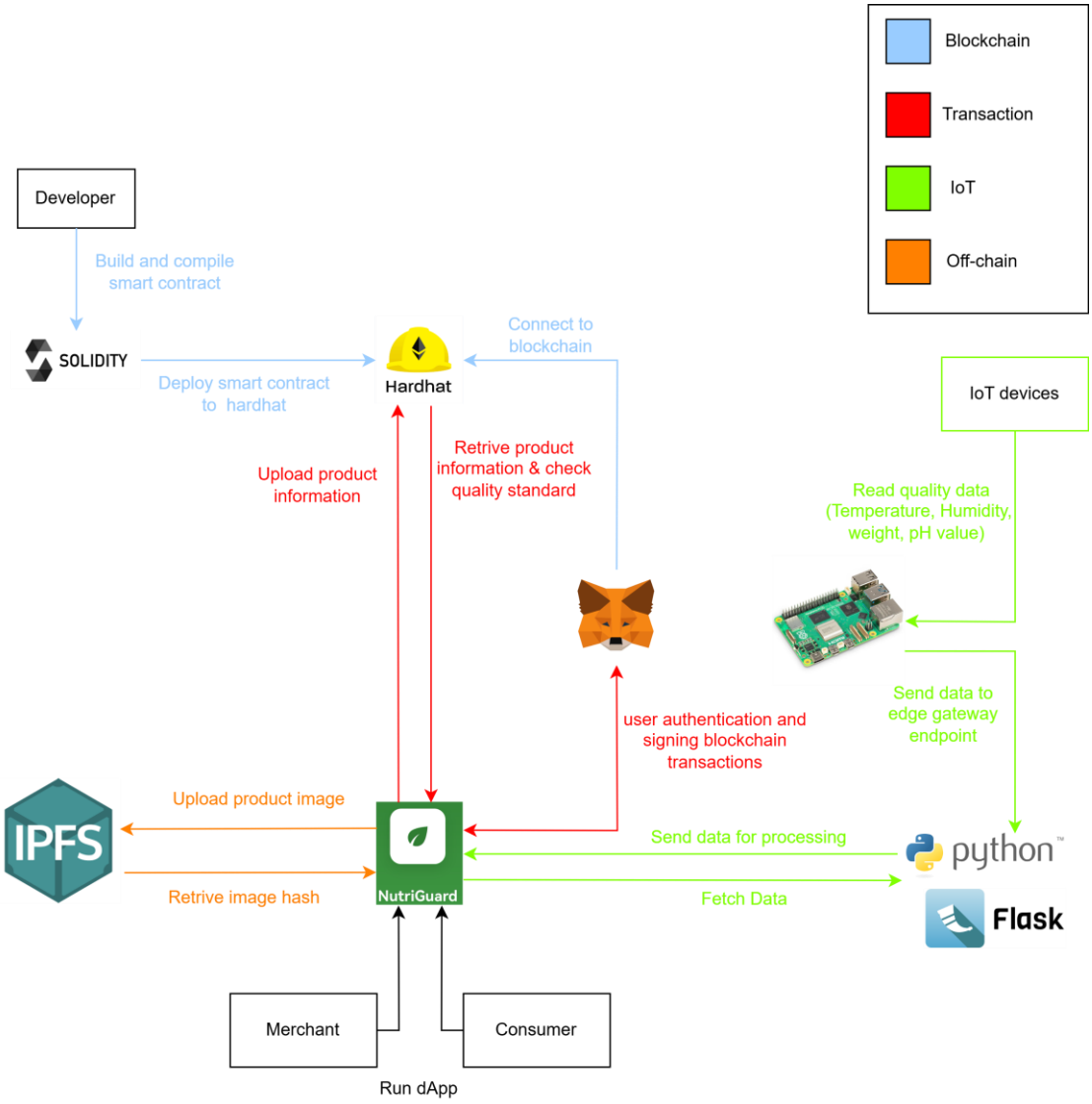


Figure 3.3 NutriGuard System Architecture Diagram

The system architecture illustrated in the diagram demonstrates how NutriGuard integrates blockchain, IoT, and decentralized storage to enable secure and transparent product traceability. Developers initiate the process by writing and compiling smart contracts using Solidity, which are then deployed and tested through Hardhat. These smart contracts define the core business logic, including product registration and quality validation. Once deployed, the system connects to the blockchain network, allowing secure interactions and data verification.

Merchants and consumers interact with the system through the NutriGuard decentralized application. Merchants upload product information and images, with images stored on IPFS to ensure immutability and decentralized access. The resulting image hash is retrieved and linked to on-chain records for verification. Consumers, on the other hand, use the same application to access product data, ensuring transparency and trust in the supply chain.

User authentication and transaction signing are handled via MetaMask, enabling secure communication with the blockchain. This ensures that all actions, such as uploading product data or retrieving quality information, are authenticated and tamper-proof. The application backend, built with Python and Flask, acts as a processing layer that handles data exchange between IoT devices, the blockchain, and the frontend.

At the edge layer, IoT devices continuously monitor environmental conditions such as temperature, humidity, weight, and pH value. These sensors send real-time data to Raspberry Pi edge gateway. which then forwards the data to the backend for processing and validation. The processed data is made accessible to the NutriGuard application and can also be recorded on-chain to enforce quality standards.

Overall, this architecture forms a cohesive ecosystem where blockchain ensures trust and immutability, IPFS provides decentralized storage, IoT devices enable real-time monitoring, and the Nutriguard dApp unifies all components into a seamless user experience for both merchants and consumers.

3.2.2 Use case Diagram and Description

Figure 3.4 Use case diagram of Nutriguard

Consumers interact with the system primarily for transparency and safety assurance. They begin by scan QR Code, which allows them to retrieve a product's blockchain-linked identity. After scanning, consumers can view product details, including ingredients, supplier certifications, and production data. They can also view registered product, which lists products associated with their wallet.

For proactive safety, consumers may register product alert to receive notifications about recalls or contamination events. The get alert message use case extend this functionality, indicating that alert notifications are triggered only when relevant events as recalls occur. Additionally, consumers can participate in the ecosystem by submitting feedback, which merchants later review and process.

Merchants are responsible for managing supply chain data and ensuring product compliance. The interaction begins with login, which include connect wallet, meaning wallet authentication is a mandatory step for accessing the system via blockchain. Once authenticated, merchants can onboard supply chain data by performing register supplier and register ingredients, where detailed information such as certifications, batch numbers, and storage conditions are recorded on-chain.

The create product use case is central to the workflow. It includes both select ingredients and define quality standard, indicating that a product cannot be created without linking ingredients and specifying HACCP-inspired thresholds. During production, merchants perform add production data, which extend fetch IoT reading. This shows that IoT sensor data such as temperature and humidity from a Raspberry Pi gateway can optionally be retrieved to assist in submitting accurate production data, but it is not strictly required.

Once the product meets all validation rules enforced by the smart contract, merchants can generate QR Code, enabling traceability for end users. In case of safety issues, merchants can initiate corrective actions through initiate recall, which include mark ingredient contaminated, ensuring that contamination marking is always part of the recall process.

Finally, merchants can handle customer interactions via review feedback, which extend mark feedback processed, meaning feedback processing is an optional follow-up step after reviewing.

3.2.3 Activity Diagram

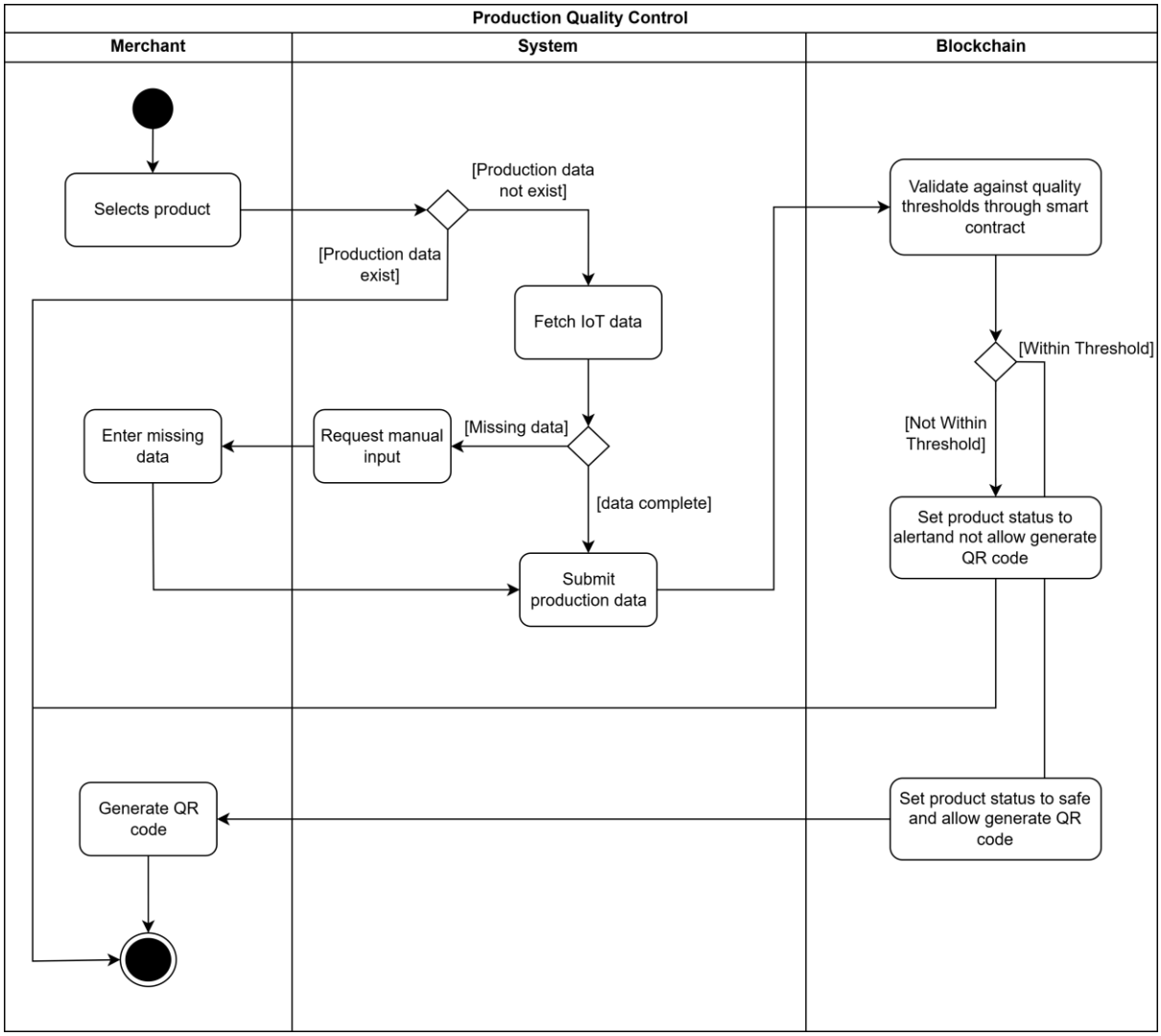


Figure 3.5 Activity diagram of Quality Control Workflow

The Production Quality Control activity begins with the Merchant selecting a product for validation. The System first checks whether production data already exists for the selected product. If the data is already available, the process terminates early, avoiding redundant submissions. Otherwise, the system proceeds to collect the necessary production data. It attempts to automatically fetch IoT readings such as temperature and humidity from the edge device. If some required data is still missing, the system prompts the merchant to manually input additional parameters, specifically weight and pH values.

Once all required data is complete, the system submits the production data to the Blockchain, where a smart contract performs validation against predefined quality thresholds. Based on the validation result, two possible outcomes occur. If the data is not within the acceptable threshold, the blockchain sets the product status to *alert* and disables QR code generation, effectively preventing the product from entering the market. Conversely, if the data meets the quality standards, the blockchain marks the product as *safe* and enables QR code generation. The merchant can then proceed to generate a QR code, which serves as a traceability link for consumers. The process ends after the product is either validated and approved or flagged as unsafe.

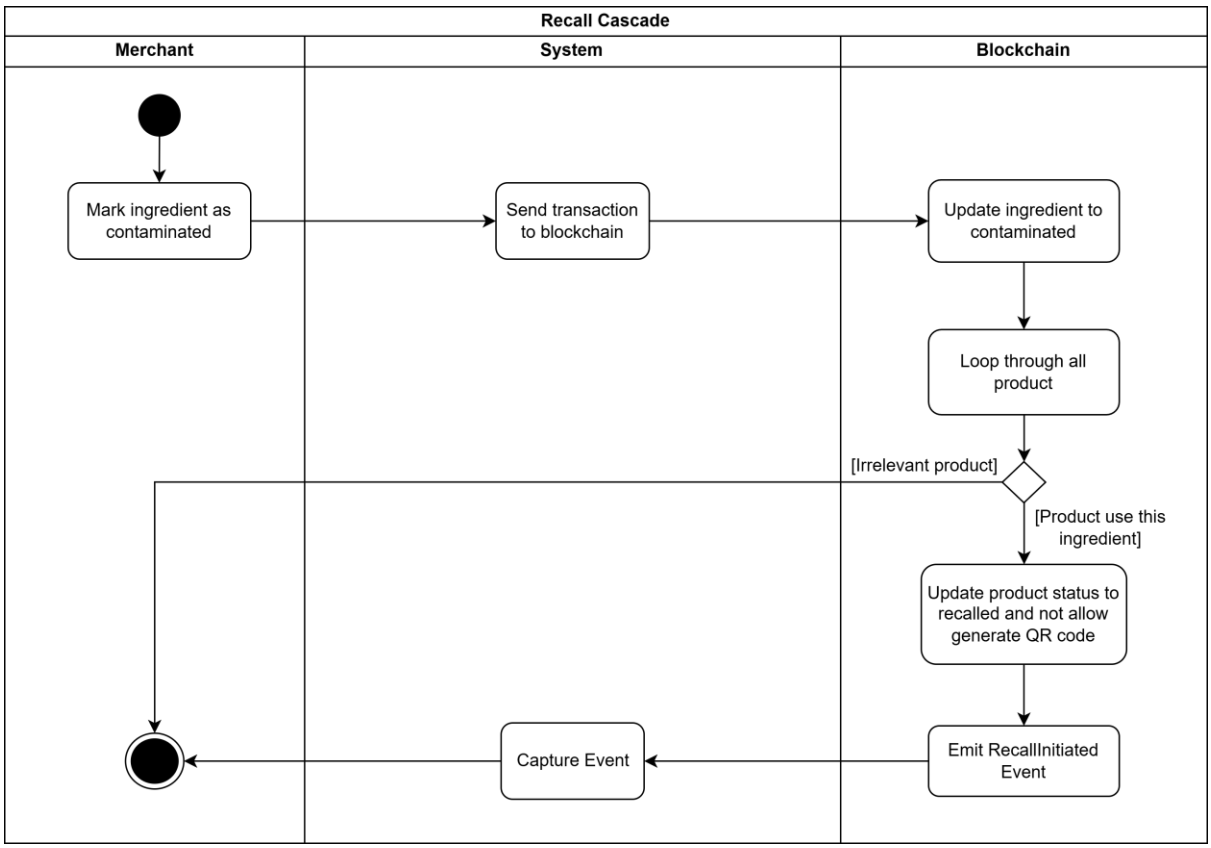


Figure 3.6 Activity diagram of Recall Cascade Workflow

The Recall Cascade activity describes how contamination or recall events propagate through the system. The process starts when a Merchant marks a specific ingredient as contaminated. This action triggers the System to send a transaction to the Blockchain, where the ingredient's status is updated accordingly.

Following this update, the blockchain initiates a loop through all registered products to determine whether any of them use the contaminated ingredient. For each product, a decision is made: if the product is unrelated, it is skipped; however, if the product

contains the affected ingredient, the blockchain updates its status to *recalled* or *contaminated*. Additionally, the product is marked as invalid and is no longer allowed to generate a QR code, ensuring it cannot be distributed further.

After processing all relevant products, the blockchain emits a Recall Initiated event. The system captures this event and makes the updated recall information available to users. This mechanism ensures that contamination information is consistently and transparently propagated across the entire supply chain. The process concludes once the recall event has been recorded and communicated.

3.3 Data Model and Smart Contract Methodology

NutriGuard uses a single Solidity smart contract named nutriguard.sol. The contract defines key entities as structs:

Entity	**Attributes**
Supplier	supplier ID, name, contact information, merchant address, creation time, active status, certifications
Ingredient	ingredient ID, name, category, UPC, supplier ID, merchant address, production date, expiry date, batch number, recall status, contamination status, storage environment, weight, IPFS hash
Product	product ID, name, description, UPC, category, status, merchant address, ingredient IDs, creation time, production date, validity status, QR eligibility, QR code hash, IPFS hash, quality rule, production data
QualityRule	minimum temperature, maximum temperature, minimum humidity, maximum humidity, minimum weight, maximum weight, minimum pH, maximum pH
ProductionData	temperature, humidity, weight, pH value, timestamp, compliance result
ConsumerFeedback	feedback ID, product ID, consumer address, feedback text, rating, timestamp, processed status
ConsumerRegistration	consumer address, product ID, email, registration time, active status

Table 3.1 Table of Entities and Attributes in Smart Contract

The smart contract design in NutriGuard defines multiple entities as on-chain data structures. Instead of treating them as independent records, the system models their relationships and lifecycle transitions through controlled smart contract functions and validation rules.

The Ingredient entity serves as the foundation of the traceability and recall mechanism.

Entity Name	Ingredient
Created By	Merchant
Storage	On chain struct
Updatable	Limited (only recall/contamination status)
Key Functions	• registerIngredient() • markIngredientContaminated()
Events Emitted	• IngredientRegistered • IngredientContaminated
Security Conditions	• Only the merchant who owns the ingredient can modify it • Contaminated ingredients cannot be reused in product creation

Table 3.2 Ingredient Entity Design Detail

The Product entity represents a composite object that links multiple ingredients and production data.

Entity Name	Product
Created By	Merchant
Storage	On chain
Updatable	State-based updates only
Key Functions	• createProduct() • submitProductionData() • generateQRCode()
Events Emitted	• ProductCreated • QCValidated • QRGenerated
Security Conditions	• Product cannot be created using contaminated or recalled ingredients

	• QR code generation is only allowed after successful quality validation • Production data can only be submitted once

Table 3.3 Product Entity Design Detail

The smart contract methodology follows the principle of minimal trusted authority. The system enforces state transitions through smart contract validation logic.

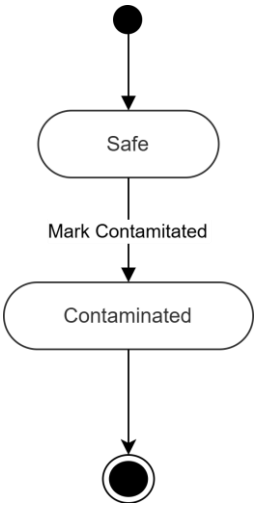


Figure 3.7 Ingredient Lifecycle

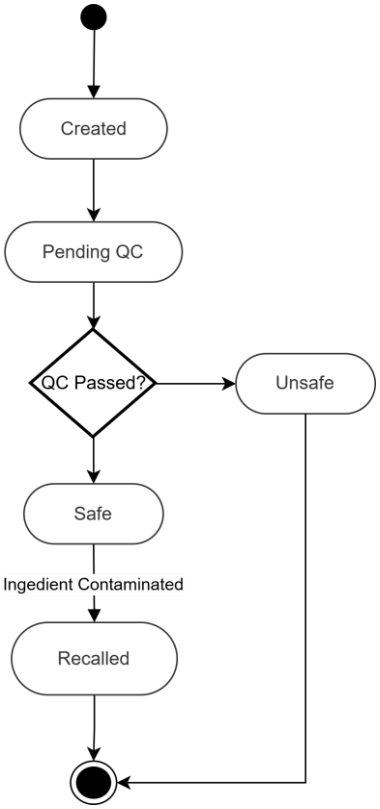


Figure 3.8 Product Lifecycle

Once records are written on chain, they cannot be silently changed. Critical validation rules are encoded in the contract:

1. A merchant can only use their own suppliers and ingredients.
2. A product cannot be created using recalled or contaminated ingredients.
3. Production data can only be submitted once per product.
4. QR code generation is blocked until production data passes quality validation.
5. Ingredient recall automatically invalidates affected products.
6. Consumer feedback can be processed only by the merchant that owns the relevant product.

3.4 IoT Edge Gateway Methodology

The IoT edge gateway in NutriGuard is designed to collect environmental data and provide it to the application layer for production quality validation. The gateway is implemented using a Raspberry Pi device connected to a DHT22 sensor and exposes RESTful APIs through a Flask-based service.

The sensor operates at a configurable sampling interval, typically set to every 2 seconds. Instead of relying on a single reading, the gateway maintains a sliding window of recent sensor values to compute a more stable average for temperature and humidity. This approach reduces noise and improves reliability of the collected data.

To ensure data freshness, a stale threshold is defined. If no valid reading is obtained within a specified duration, the data is marked as stale. The system classifies sensor output into three categories: ok, stale, and error, allowing the application to determine whether the data can be safely used.

The gateway exposes multiple API endpoints for data access:

- /sensor returns raw sensor readings
- /latest returns processed and validated data

In cases where the sensor fails or returns invalid data, the gateway responds with an error status. The Flutter application verifies the data quality before auto-filling production parameters, ensuring that unreliable readings are not used in quality validation.

To improve system robustness, a manual fallback mechanism is provided. If the IoT gateway is unavailable or the data is marked as stale, the merchant can manually input

production data such as weight and pH values. This ensures that system operation is not blocked by hardware or network issues.

When the IoT gateway is offline, the system continues to function without automated sensor input. However, in such cases, production validation relies entirely on manually entered data, which may reduce accuracy. This design reflects a hybrid approach, balancing automation with operational reliability in a prototype environment.

3.5 Timeline

The project timeline is structured as follows:

Name	Begin date	End date	Duration
Confirm FYP2 scope, align FYP1 gaps, finalize technical stack	02/02/2026	06/02/2026	5
Review and refactor smart contract baseline	09/02/2026	13/02/2026	5
Stable merchant CRUD on local chain	16/02/2026	25/02/2026	8
Implement production data submission and validation logic	26/02/2026	06/03/2026	7
Implement recall cascade and affected-product status updates	09/03/2026	16/03/2026	6
Integrate and testing consumer functions	17/03/2026	24/03/2026	6
Build and verify Raspberry Pi and sensor reading scripts	25/03/2026	31/03/2026	5
Implement Flask edge gateway and service setup	01/04/2026	03/04/2026	3
Integrate Flutter IoT auto-fill button and sensor status handling	06/04/2026	09/04/2026	4
Execute functional and integration testing	10/04/2026	15/04/2026	4
Debug and collect evidence and screenshots	16/04/2026	20/04/2026	3
Final evaluation write-up, report polishing	20/04/2026	28/04/2026	7

Figure 3.9 Timeline (Part 1)

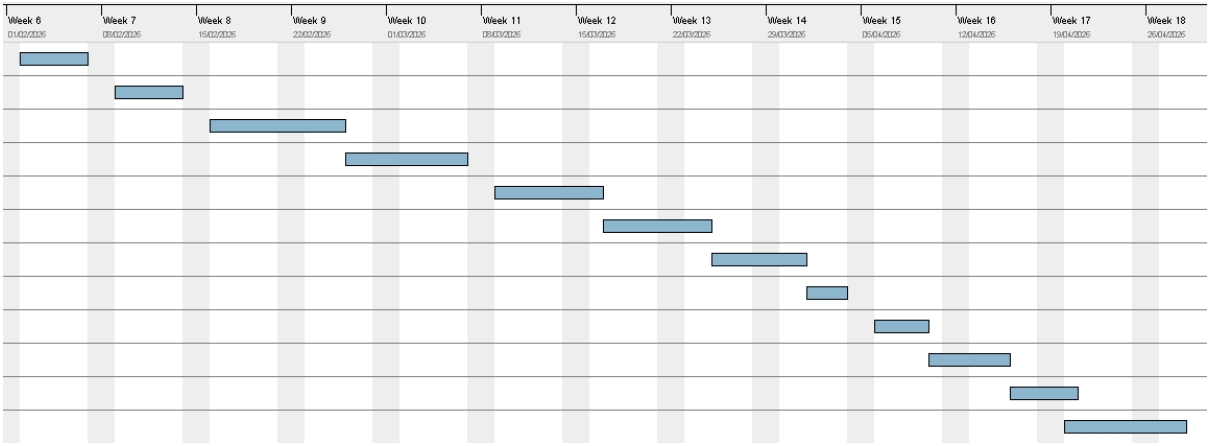


Figure 3.10 Timeline (Part 2)

Chapter 4
System Design

4.1 System Overview and Block Diagram

The NutriGuard system integrates blockchain, IoT, and mobile applications to ensure transparent and reliable food traceability. The system consists of three main components: the mobile applications (merchant and consumer), the blockchain smart contract layer, and the IoT gateway.

A key feature of the system is the recall cascade mechanism, which ensures that when an ingredient is marked as contaminated, all related products are automatically identified and updated. The recall flow can be described as:

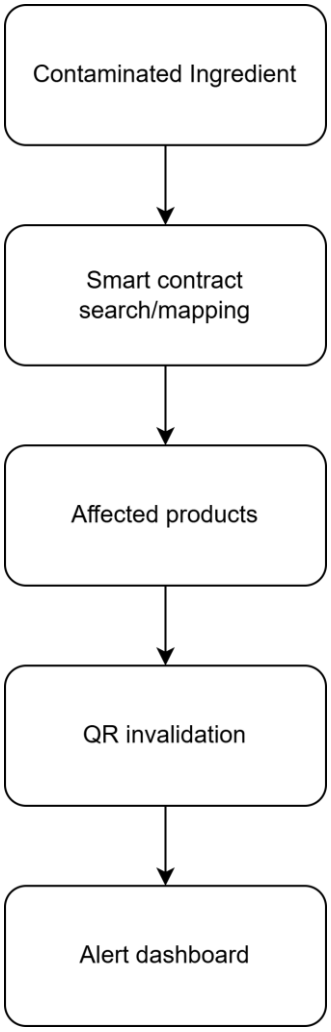


Figure 4.1 Recall Cascade Path

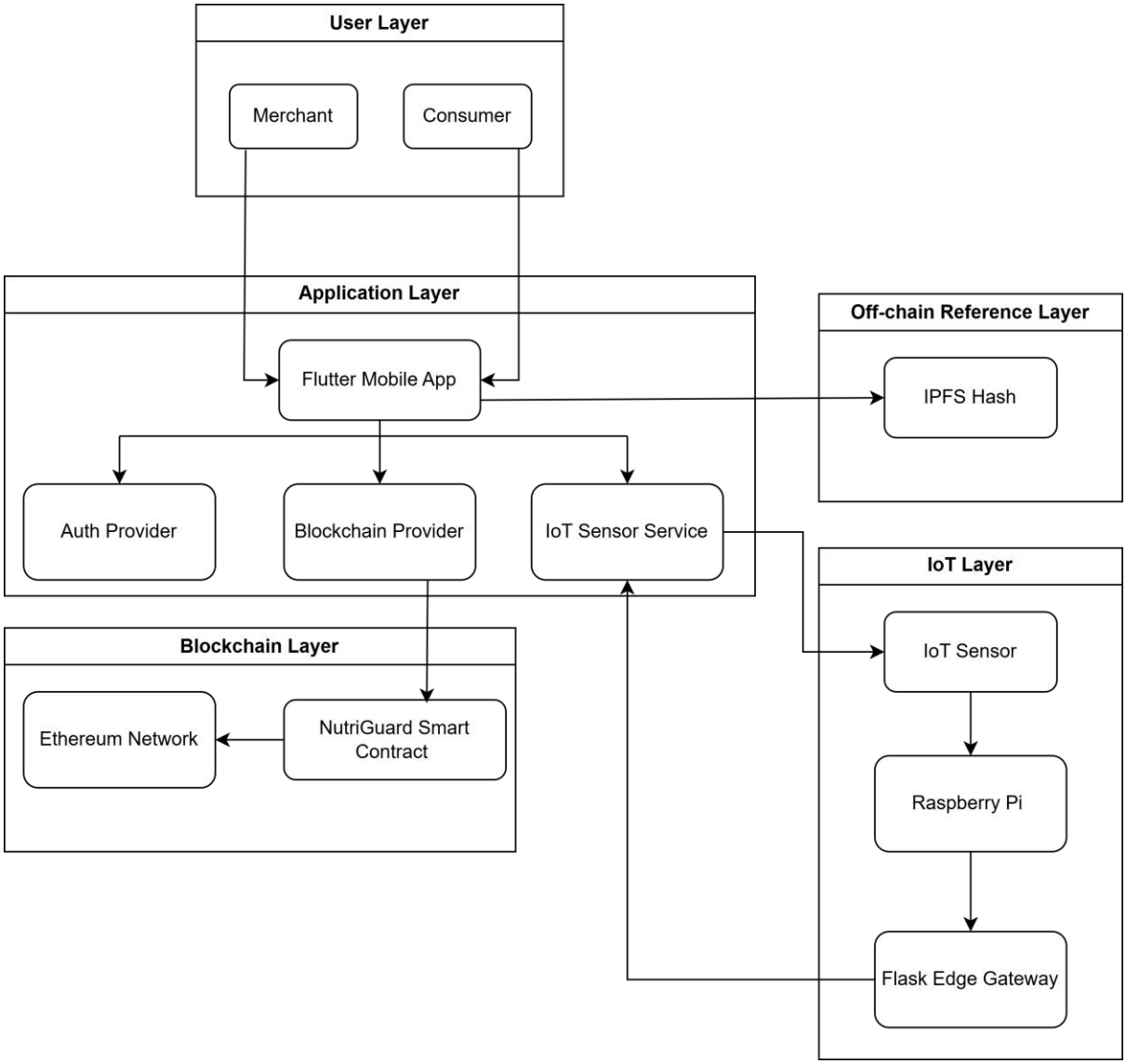


Figure 4.2 System Block Diagram of NutriGuard

The system block diagram of NutriGuard illustrates a layered architecture that integrates blockchain, mobile applications, IoT devices, and decentralized storage to provide a transparent and reliable food traceability solution.

At the user layer, two primary actors interact with the system: merchants and consumers. Merchants are responsible for managing suppliers, ingredients, and products, while consumers access product information and traceability data. Both users interact with the system through the mobile interface provided in the application layer.

The application layer is built using a Flutter mobile application, which serves as the central interface connecting all system components. Within this layer, three main

modules are defined. The auth provider handles user authentication and wallet connection, enabling secure access to blockchain functionalities. The blockchain provider manages communication with the smart contract, including sending transactions and retrieving on-chain data. The IoT sensor service acts as a bridge between the application and physical devices, collecting environmental data such as temperature and humidity. In this project prototype, IoT data is generated using a DHT22 sensor, which measures temperature and humidity as part of the production monitoring process.

The blockchain layer consists of the NutriGuard smart contract deployed on a Hardhat local network, which simulates an Ethereum environment for development and testing. This layer is responsible for enforcing business logic, such as validating production data against predefined quality thresholds, managing product and ingredient records, and handling recall events. The use of Hardhat allows efficient contract deployment, debugging, and local transaction testing without requiring a public blockchain.

The IoT layer includes the physical sensing and data acquisition components. The DHT22 sensor captures real-time environmental data, which is processed by a Raspberry Pi acting as an edge device. The data is then forwarded through a Flask-based edge gateway, which exposes APIs for the application layer to retrieve sensor readings. This design ensures that real-world environmental conditions can be integrated into the blockchain validation process.

The off-chain reference layer utilizes IPFS to store large or unstructured data such as product images and certificates. Instead of storing files directly on-chain, only the IPFS hash is recorded in the smart contract, ensuring data integrity while minimizing blockchain storage costs.

Overall, this architecture separates concerns across multiple layers while maintaining seamless integration. The Flutter application unifies user interaction, blockchain communication, IoT data acquisition, and off-chain storage into a cohesive system. By combining a Hardhat-based blockchain environment with DHT22 sensor data, the

NutriGuard prototype demonstrates a practical and scalable approach to achieving transparency, traceability, and quality assurance in the food supply chain.

4.2 System Component Specification

4.2.1 Hardware Component Specifications

Component	Specification	Purpose
Development Laptop	Windows development machine with Node.js, Flutter SDK, and Python	Smart contract, mobile app, and IoT gateway development
Mobile Device	Android-compatible target	Running Flutter merchant and consumer application
Raspberry Pi 3 Model B+	1GB RAM, 40-pin GPIO header, WiFi/Ethernet	Edge gateway for sensor acquisition
DHT22 Sensor	Temperature range -40-125°C, humidity range 0-100% RH	Temperature and humidity monitoring
MicroSD Card	16GB or above	Raspberry Pi OS and gateway scripts
Jumper Wires	Female-to-female wires	Sensor connection to GPIO

Table 4.1 Hardware Specifications

4.2.2 Software and Technology Stack

Layer	Technology	Version/Notes	Purpose
Smart Contract	Solidity	0.8.28	Contract implementation
Blockchain Framework	Hardhat	2.26.3	Local blockchain, compilation, deployment
Smart Contract Library	OpenZeppelin Contracts	5.4.0	Ownable, ReentrancyGuard
Mobile Application Framework	Flutter	SDK 3.x	Cross-platform mobile app

Mobile Application Language	Dart	SDK 3.8.1	Application logic
Blockchain Client	Web3dart	2.7.3	Flutter-to-Ethereum interaction
Wallet	Metamask	7.73.2	Authentication and transaction signing
IoT Runtime	Python	3.11+	Sensor gateway scripts
Edge API	Flask+flask-cors	Flask 3.x	REST API endpoints
Sensor Driver	adafruit-circuitpython-dht	3.7+	DHT22 reading
Direct Chain Agent	Web3.py	6.x	Raspberry Pi direct submission

Table 4.2 Software Used and Technology Stack

4.2.3 Sensor Data JSON Contract

Field	**Type**	**Description**
device_id	String	Unique sensor gateway identifier
timestamp	Integer	Unix timestamp generated by Raspberry Pi
temperature_c	Double/null	Sliding-window average temperature in Celsius
humidity_pct	Double/null	Sliding-window average humidity percentage
sample_count	Integer	Number of valid samples used in average
quality	String	ok, stale, or error

Table 4.3 Field of JSON Data

Figure 4.3 Example of JSON Output

4.2.4 Smart Contract Functions

Function	Actor	Input	State Updated	Event Emitted	Failure Condition
registerSupplier	Merchant	supplier data	supplier stored	SupplierRegistered	duplicate
registerIngredient	Merchant	ingredient data	ingredient stored	IngredientRegistered	invalid input
createProduct	Merchant	ingredient IDs	product created	ProductCreated	ingredient recalled
submitProduction	Merchant	sensor data	QC state updated	QCUpdated	duplicate submission
generateQR	Merchant	product ID	QR activated	QRGenerated	QC not passed
recallIngredient	Merchant	ingredient ID	ingredient flagged	IngredientRecalled	already recalled

Table 4.4 Functions of Smart Contract

4.2.5 Access Control and Security Modifiers

The system enforces strict access control using smart contract modifiers:

1. OnlyMerchant
2. OnlyProductOwner
3. Prevent duplicate production submission
4. Prevent product creation using recalled ingredients
5. Prevent QR generation before QC approval
6. Restrict feedback processing to authorized users

4.3 System Component Interaction Operations

4.3.1 Supplier and Ingredient Registration

The merchant first registers supplier details. After that, the merchant registers ingredients linked to the supplier. Ingredient information includes name, category, UPC, production date, expiry date, batch number, storage temperature range, storage humidity range, weight, and IPFS hash field.

4.3.2 Product Creation and HACCP Rule Definition

The merchant creates a product by selecting one or more existing ingredients. The smart contract verifies that each ingredient exists, belongs to the merchant, and is not recalled or contaminated. The merchant also defines quality thresholds, including temperature, humidity, weight, and pH range. pH validation is applied only to beverage products.

4.3.3 IoT-Assisted Production Data Submission

For products without production data, the merchant opens the quality-control screen. The merchant can click the IoT reading button to fetch temperature and humidity from the Raspberry Pi gateway. The app validates whether the sensor data has *quality = ok*. If the reading is valid, temperature and humidity fields are auto filled. Weight and pH are manually entered in the current implementation. The merchant then submits the data to the smart contract.

4.3.4 QR Code Generation and Consumer Verification

If production data is compliant, the contract sets canGenerateQR to true. The merchant can then generate a QR code. Consumers scan the QR code to retrieve product details, ingredient list, production status, and safety status.

4.3.5 Recall Cascade

When a merchant detects contamination in an ingredient, the merchant can initiate recall or mark the ingredient contaminated. The contract updates the ingredient status and scans all products to find those that use the ingredient. Affected products are marked as contaminated, invalidated, and blocked from QR generation.

4.3.6 Feedback Processing

Consumers can submit feedback and ratings for products. Merchants can review feedback associated with their own products and mark it as processed. This supports two-way communication between consumers and merchants.

4.4 Security and Privacy Design

NutriGuard is a research prototype. Security and privacy must therefore be described in two layers: what the design intends to achieve and what remains insufficient for production. The subsections below address privacy-sensitive data, data placement, IPFS semantics, deployment assumptions, smart-contract hardening, key management, and trust in IoT data.

4.4.1 Storing Raw Email On-Chain

Email addresses are personally identifiable information. On a public or consortium blockchain, any data written to contract storage is world-readable, permanent, and cannot be erased without a chain fork or migration. Consequences include:

- Privacy breach: Third parties can scrape registered emails and link them to wallet addresses and purchased products.
- Regulatory risk: Laws such as the GDPR treat email as personal data; immutability conflicts with data-minimization and erasure principles unless carefully engineered.
- Phishing and spam: Public email lists increase exposure to targeted attacks pretending to be the merchant or the platform.
- No confidentiality: On-chain storage is not encrypted at the application layer; anyone with RPC access sees the plaintext string.

The current prototype uses registerForProductAlerts to store email strings for demonstration. A production system should not persist raw email in contract storage.

4.4.2 On-Chain and Off-Chain Data

A practical partition balances immutability and auditability (on-chain) with privacy, size, and cost (off-chain).

Category	Prefer on-chain	Prefer off-chain

Supply-chain facts	Supplier/ingredient/product IDs, batch numbers, timestamps, recall flags, compliance outcomes, hashes of documents	Full PDF certificates, high-resolution images, long narrative reports
Quality control	Submitted production values used for gating, compliance boolean, block timestamp	Raw sensor logs at high frequency, calibration certificates, operator notes
Consumer engagement	Wallet address, product ID binding, commitment to alert subscription (e.g. hash)	Email, phone number, home address
QR / provenance	QR content hash or product ID reference	Marketing copy, rich media not needed for verification

Table 4.5 On-Chain vs Off-Chain Data

4.4.3 Limitations of IPFS Hash Fields

The contract exposes IPFS hash (CID) fields for ingredients and products. However, storing a hash on-chain alone does not guarantee several critical properties. First, availability is not ensured, as files may become unpinned, garbage-collected, or lost if no pinning service or replication mechanism is maintained. Second, the authenticity of uploaded content is not inherently verified, since any party capable of interacting with the contract could potentially store arbitrary data unless uploads are restricted through validated identities and controlled workflows. Third, while IPFS uses content addressing, immutability applies only to the specific file tied to a CID as any modification results in a new CID, requiring corresponding updates on-chain if documents are replaced or updated. Finally, privacy is not guaranteed, as IPFS operates largely as a public network where unencrypted content may be accessible or discoverable. Therefore, production deployments should incorporate reliable pinning services, access-controlled gateways, and well-defined governance policies to regulate who can associate specific CIDs with on-chain entities.

4.4.4 Limitations of Local Hardhat Network

The prototype is demonstrated primarily on Hardhat's local Ethereum node, an environment that differs significantly from mainnet or public testnets in several important ways. First, the state is ephemeral: restarting the node without proper persistence resets all accounts, balances, and contract state unless explicitly scripted, which limits its usefulness for long-term or realistic testing. Second, identity management is trivial, as the pre-funded test accounts are publicly known; while this poses no real financial risk in a development setting, it can give a misleading impression about private key security and proper key management practices. Third, network conditions in Hardhat are highly idealized, latency is negligible, transaction finality is immediate, and RPC reliability is near perfect, unlike real-world environments where users may interact via mobile devices on unstable cellular networks. As such, while findings from Hardhat are valuable for validating contract logic and application workflows, they do not demonstrate robustness or security under adversarial or realistic network conditions. At this stage, Hardhat results should be explicitly framed as functional verification only, with further testing recommended on networks such as Sepolia testnet or suitable Layer-2 testnets to better approximate real-world threat models and user experience.

4.4.5 Reentracy Protection

The contract inherits OpenZeppelin's ReentrancyGuard. In Solidity, reentrancy becomes dangerous when a function performs an external call to untrusted code before finishing state updates, violating checks-effects-interactions. In the current NutriGuard.sol source, no function applies the nonReentrant modifier, so the guard is prepared but not actively enforcing reentrancy safety at runtime.

NutriGuard's present flows are largely internal state updates without payable calls to arbitrary contracts, which limits classical reentrancy exposure. Still, future upgrades that add token transfers, external hooks, or delegatecall should apply nonReentrant on those entry points and follow checks-effects-interactions strictly.

4.4.6 Access Control

Access control in the prototype is implemented through a combination of OpenZeppelin's Ownable pattern for contract-level authority and explicit per-function checks, such as require(products[_productId].merchant == msg.sender), to restrict merchant-only operations. Consumer-specific interactions similarly depend on msg.sender as the registering wallet, ensuring that actions are tied directly to the caller's address. While this approach is sufficient for a prototype, earlier designs that relied on broad registerUser roles introduce ambiguity and potential risks. In a production setting, adopting a more explicit role-based access control model such as OpenZeppelin's AccessControl with clearly defined roles like Merchant and Consumer provides stronger guarantees and helps prevent accidental privilege escalation as the contract evolves and new functions are introduced.

A key principle for secure design is that every state-changing function must clearly define three aspects: who is permitted to call the function, on whose behalf the action is performed, and which asset or record is being modified. Enforcing this discipline ensures that permissions remain transparent, auditable, and resistant to misuse.

4.4.7 Private Key and Wallet Assumptions

The current demonstration relies on Hardhat preset accounts embedded directly in the application configuration for convenience and speed. While this approach is acceptable in a coursework or prototype setting, it is fundamentally insecure for any environment involving real assets. Embedding private keys in a repository or distributed APK means that anyone with access to these files can impersonate merchants or consumers. Similarly, components such as the RPi chain_agent.py that load a MERCHANT_PRIVATE_KEY from a .env file introduce significant risk. If the device is stolen or the file is exposed, the associated account can be fully compromised.

In a production environment, these assumptions must be replaced with stronger key management practices. Users should operate through self-controlled wallets, such as MetaMask, hardware wallets, or secure mobile wallet SDKs, ensuring that private keys are never hardcoded or exposed within application code. IoT devices should avoid storing long-lived private keys unless they are protected by dedicated hardware security

mechanisms such as secure enclaves or hardware security modules. Additionally, systems should implement key rotation policies and maintain clear separation of responsibilities by using distinct keys for contract deployment, operational tasks, and end-user interactions. These measures collectively reduce the attack surface and align the system with standard security practices for blockchain-based applications.

4.4.8 Oracle and Sensor Trust

Blockchain smart contracts cannot directly access external data sources such as HTTP APIs or physical sensors like GPIO devices. As a result, NutriGuard depends on trusted off-chain components to relay real-world data onto the chain, introducing a critical trust boundary. For instance, the DHT22 sensor used for temperature and humidity measurements has limited accuracy and can be easily manipulated by heating, cooling, or even disconnecting it. The Flask-based gateway that exposes this data can also be compromised, potentially returning spoofed JSON responses. On the client side, the Flutter application signs and submits transactions; however, a modified or malicious version of the app could inject fabricated readings even if the gateway itself is trustworthy.

This situation reflects the well-known oracle problem: the blockchain can only verify what a signing key attests to, not whether the underlying data is truthful. To strengthen trust in such systems, several mitigation strategies can be applied. One approach is signed telemetry, where sensor readings are cryptographically signed by a device-specific key and verified before being accepted either off-chain or through a dedicated oracle verification mechanism. Another method is multi-source data fusion, where readings from edge devices are cross-checked against independent logging systems to detect anomalies or tampering. More advanced approaches include introducing economic or reputation-based staking models for data providers, incentivizing honest reporting and penalizing malicious behavior. Together, these measures help reduce reliance on any single point of trust and improve the integrity of data entering the blockchain.

4.4.9 Threat Model Analysis

Threat	Risk	Mitigation	Limitation
Fake merchant	Invalid data entry	Wallet authentication	No identity verification
Sensor tampering	Incorrect QC results	Manual fallback	No trusted oracle
Recalled ingredient reuse	Unsafe product creation	Smart contract validation	Depends on logic completeness
QR reuse after recall	Consumer safety risk	QR invalidation	Cached QR may exist
Email exposure	Privacy issue	Avoid on-chain storage	Not fully implemented
Smart contract bug	System failure	Testing & validation	No formal verification

Table 4.6 Threat Model Table

Chapter 5

System Implementation

5.1 Hardware Setup

The IoT prototype uses a Raspberry Pi 3B+ and DHT22 sensor. The DHT22 module is connected to the Raspberry Pi GPIO header as follows:

DHT22 Pin	Raspberry Pi Physical Pin	Function
VCC	Pin 1	3.3V Power
Data	Pin 7	GPIO4 data line
GND	Pin 6	Ground

Table 5.1 Pin Used In Raspberry Pi

The DHT22 was selected because it is inexpensive and simple to integrate for a proof-of-concept. Its lower accuracy compared with SHT31 or SHT85 is recognized as a limitation. The sensor is sufficient to demonstrate the end-to-end process of environmental data acquisition, edge gateway processing, and smart contract validation.

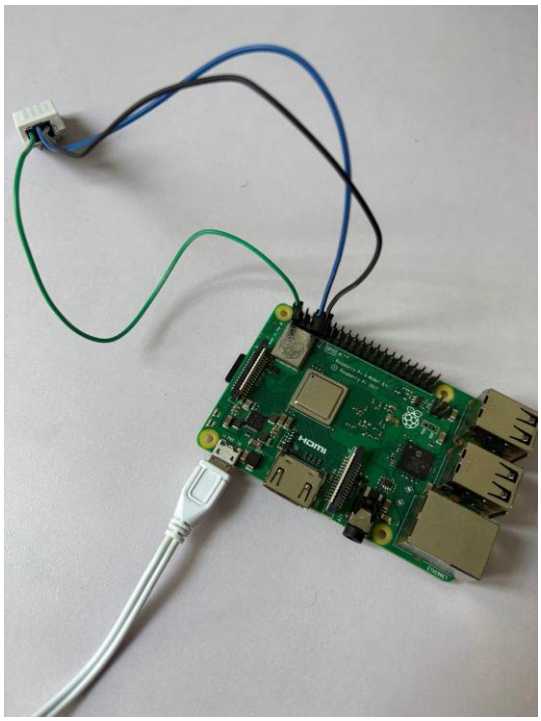


Figure 5.1 Raspberry Pi 3B+ and DHT22 wiring setup

5.2 Software Setup

5.2.1 Blockchain Environment

The blockchain environment is implemented using Hardhat. The development steps are:

1. Install Node.js and npm.
2. Install project dependencies in the Blockchain directory.
3. Compile the Solidity contract with Hardhat.
4. Start the Hardhat local node with network access enabled.
5. Deploy NutriGuard.sol to the local network.
6. Copy the deployed contract address into the Flutter AppConfig

```
require("@nomicfoundation/hardhat-toolbox");

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: {
    version: "0.8.28",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
      },
      viaIR: true,
    },
  },
  networks: {
    hardhat: {
      chainId: 1337,
    },
    localhost: {
      url: "http://192.168.0.11:8545",
    },
  },
  paths: {
    sources: "./contracts",
    tests: "./test",
    cache: "./cache",
    artifacts: "./artifacts",
  },
};
```

Figure 5.2 hardhat.config.js

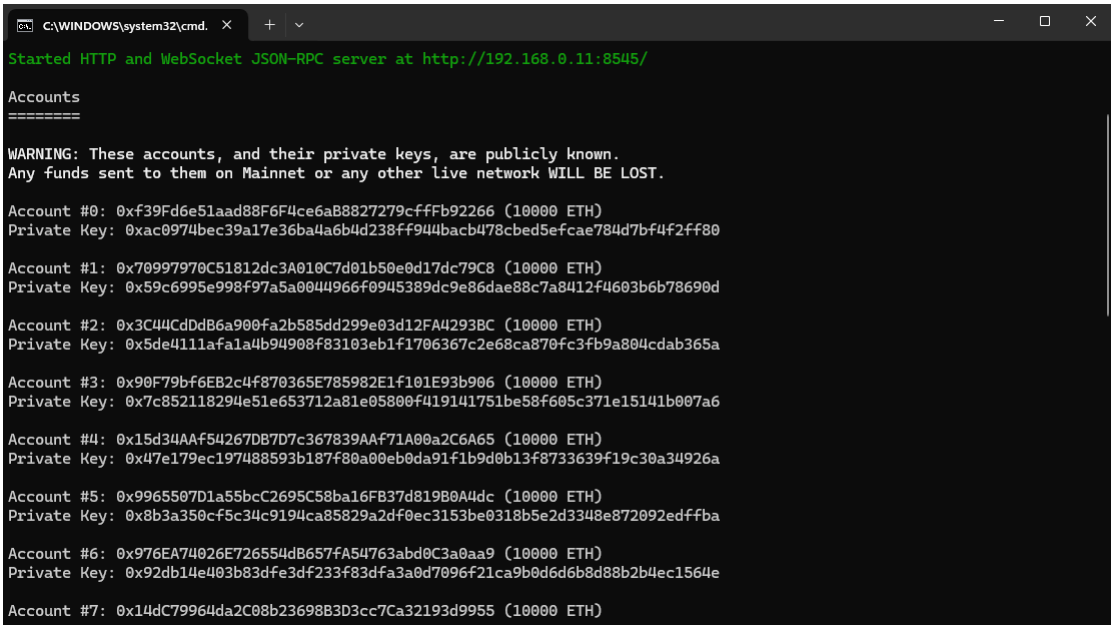


Figure 5.3 Hardhat local blockchain network

```
D:\nutriguard\Blockchain>npx hardhat run scripts/deploy.js --network localhost
Deploying contracts with the account: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
Account balance: 10000000000000000000000
NutriGuard contract deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Contract with simplified roles (Merchant/Consumer) deployed successfully!
Deployment completed successfully!
Contract addresses saved to nutri_guard/lib/config/contract_addresses.json
```

Figure 5.4 Smart Contract Deployment Successful

5.2.2 Flutter Application Environment

The Flutter mobile application is implemented in the *nutri_guard* directory. It uses Provider for state management, *GoRouter* for navigation, *web3dart* for blockchain interaction, and *mobile_scanner / qr_flutter* for QR scanning and generation.

```
D:\nutriguard\nutri_guard>flutter pub get
Resolving dependencies... (8.4s)
Downloading packages... (2.3s)
  appcheck 1.5.4+1 (1.7.0 available)
  async 2.13.0 (2.13.1 available)
  cached_network_image 3.4.0 (3.4.1 available)
  cached_network_image_web 1.3.0 (1.3.1 available)
  connectivity_plus 6.1.5 (7.1.1 available)
  connectivity_plus_platform_interface 2.0.1 (2.1.0 available)
  cross_file 0.3.4+2 (0.3.5+2 available)
  crypto 3.0.6 (3.0.7 available)
  cryptography 2.7.0 (2.9.0 available)
  cupertino_icons 1.0.8 (1.0.9 available)
  dbus 0.7.11 (0.7.12 available)
  eip55 1.0.2 (1.0.3 available)
  event 2.1.2 (3.1.0 available)
  ffi 2.1.4 (2.2.0 available)
  flutter_lints 5.0.0 (6.0.0 available)
  flutter_svg 2.0.13 (2.2.4 available)
  freezed_annotation 2.4.4 (3.1.0 available)
  go_router 12.1.3 (17.2.2 available)
  http 1.5.0 (1.6.0 available)
  intl 0.19.0 (0.20.2 available)
  js 0.6.7 (0.7.2 available)
  json_annotation 4.9.0 (4.11.0 available)
  json_rpc_2 3.0.3 (4.1.0 available)
  lints 5.1.1 (6.1.0 available)
  logger 2.6.1 (2.7.0 available)
```

Figure 5.5 Run flutter pub get command

```
Launching lib\main.dart on YAL L21 in debug mode...
Running Gradle task 'assembleDebug'...
√ Built build\app\outputs\flutter-apk\app-debug.apk
Installing build\app\outputs\flutter-apk\app-debug.apk...
D/FlutterJNI(15224): Beginning load of flutter...
D/FlutterJNI(15224): flutter (null) was loaded normally!
I/flutter (15224): [IMPORTANT:flutter/shell/platform/android/android_context_gl_impeller.cc(104)] Using the Impeller rendering backend (OpenGLES).
D/FlutterRenderer(15224): Width is zero. 0,0
D/FlutterRenderer(15224): Width is zero. 0,0
D/FlutterJNI(15224): Sending viewport metrics to the engine.
D/FlutterJNI(15224): Sending viewport metrics to the engine.
Debug service listening on ws://127.0.0.1:63838/-4XeSóunw7k=/ws
Syncing files to device YAL L21...
```

Figure 5.6 flutter run and Successfully Build Application

5.2.3 Raspberry Pi Edge Gateway Environment

The Raspberry Pi edge gateway is implemented in the *nutriguard_iot* directory. It requires Python dependencies such as Flask, flask-cors, adafruit-circuitpython-dht, adafruit-blinka, web3, and python-dotenv.

```
(.venv) pi@pi-home:~/nutriguard/nutriguard_iot $ python edge_gateway.py
2026-04-28 11:53:55,253 | INFO | edge_gateway | sampler start: device_id=rpi-nut
riguard-01 gpio=D4 interval=2.0s window=20s mock=False
2026-04-28 11:53:55,253 | INFO | edge_gateway | edge gateway listening on 0.0.0.
0:5000
 * Serving Flask app 'edge_gateway'
 * Debug mode: off
2026-04-28 11:53:55,314 | INFO | werkzeug | WARNING: This is a development serve
r. Do not use it in a production deployment. Use a production WSGI server instea
d.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.20.10.2:5000
2026-04-28 11:53:55,315 | INFO | werkzeug | Press CTRL+C to quit
2026-04-28 11:53:55,605 | INFO | edge_gateway | sample ok t=23.6 h=47.2
2026-04-28 11:54:02,662 | INFO | edge_gateway | sample ok t=23.6 h=47.1
2026-04-28 11:54:05,015 | INFO | edge_gateway | sample ok t=23.6 h=47.0
2026-04-28 11:54:07,367 | INFO | edge_gateway | sample ok t=23.6 h=47.0
```

Figure 5.7 Start the IoT Edge Gateway in Raspberry Pi

5.3 Setting and Configuration

File	Configuration	Description
app_config.dart	Contract address	Deployed NutriGuard smart contract address
	Ethereum RPC URL	Hardhat RPC endpoint
	IoT gateway URL	Raspberry Pi Flask gateway URL
.env	Edge gateway settings	Device ID, port, GPIO, RPC settings
sync_abi.ps1	ABI sync	Copies smart contract ABI for Python web3 agent

Table 5.2 Summary of Configuration and File Name

```dart
class AppConfig {
  static const String appName = 'NutriGuard';
  static const String appVersion = '1.0.0';

  // Blockchain Configuration
  static const String ethereumChainId = '1337'; // Hardhat Local network
  static const String ethereumRpcUrl = 'http://172.20.10.4:8545'; //172.20.10.4
  static const String sepoliaRpcUrl = 'https://sepolia.infura.io/v3/YOUR_INFURA_KEY';

  // IoT Edge Gateway (Raspberry Pi + DHT11)
  // Raspberry Pi running Flask edge gateway address, needs to be in the same network as the App phone/simulator
  static const String iotGatewayBaseUrl = 'http://172.20.10.2:5000';
  static const Duration iotFetchTimeout = Duration(seconds: 5);

  // Contract Addresses (will be updated after deployment)
  static const String nutriGuardContractAddress = '0x5FbDB2315678afecb367f032d93F642f64180aa3';
```

Figure 5.8 app_config.dart

```
# ──────────── IoT Edge Gateway Configuration (edge_gateway.py) ────────────
DEVICE_ID=rpi-nutriguard-01
PORT=5000
SAMPLE_INTERVAL_S=2
WINDOW_SECONDS=20
DHT_GPIO=4
LOG_LEVEL=INFO

# ──────────── Blockchain Direct Connection to Chain (chain_agent.py, M3 enabled)
# Hardhat node (development machine IP, e.g. 192.168.1.100)
ETH_RPC_URL=http://172.20.10.4:8545
CHAIN_ID=1337
# Update to the latest contract address after deployment
CONTRACT_ADDRESS=0x5FbDB2315678afecb367f032d93F642f64180aa3
```

Figure 5.9 .env file

```
param(
    [Parameter(Mandatory = $true)]
    [string]$RpiHost,

    [string]$RpiPath = "/home/pi/nutriguard_iot"
)

$ErrorActionPreference = "Stop"

$repoRoot = Split-Path -Parent $PSScriptRoot | Split-Path -Parent
$abiSource = Join-Path $repoRoot "Blockchain\artifacts\contracts\NutriGuard.sol\NutriGuard.json"

if (-not (Test-Path $abiSource)) {
    Write-Error "Cannot find ABI file: $abiSource"
    exit 1
}

$remotePath = "{0}:{1}/abi/NutriGuard.json" -f $RpiHost, $RpiPath

Write-Host ("Syncing ABI to {0}:{1}/abi/..." -f $RpiHost, $RpiPath)

scp "$abiSource" "$remotePath"

if ($LASTEXITCODE -eq 0) {
    Write-Host "ABI sync completed"
} else {
    Write-Error "scp failed, please check SSH configuration"
    exit 1
}
```

Figure 5.10 sync_abi.ps1

5.4 System Operation

A complete system operation scenario is as follows:

1. Merchant registers supplier
2. Merchant registers ingredient
3. Merchant creates product
4. IoT data is retrieved from gateway
5. Smart contract validates production data
6. Product passes QC and QR is generated
7. Consumer scans QR and verifies product
8. Merchant initiates recall on ingredient
9. Smart contract updates affected products
10. QR becomes invalid and alert status is shown

5.4.1 Merchant Login

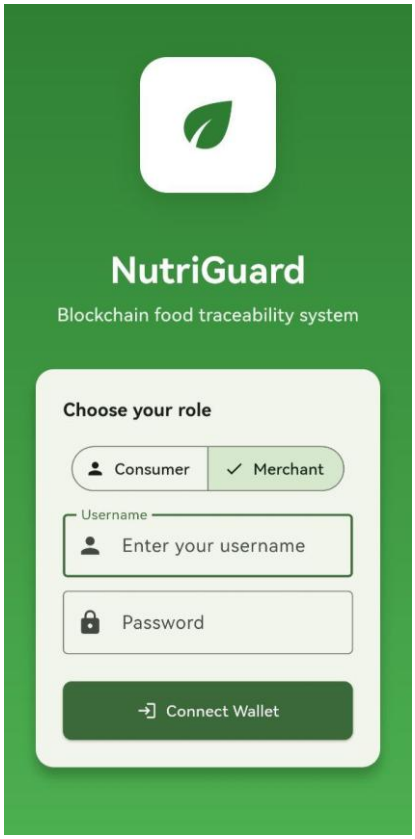


Figure 5.11 Merchant Login

Figure 5.11 shows the main page of the system, where users can select either the Consumer or Merchant role for login. If the Merchant role is chosen, the system

requires the user to enter a username and password. Once the credentials are verified, the system proceeds to establish a connection with the user's decentralized wallet.

5.4.2 Merchant Dashboard

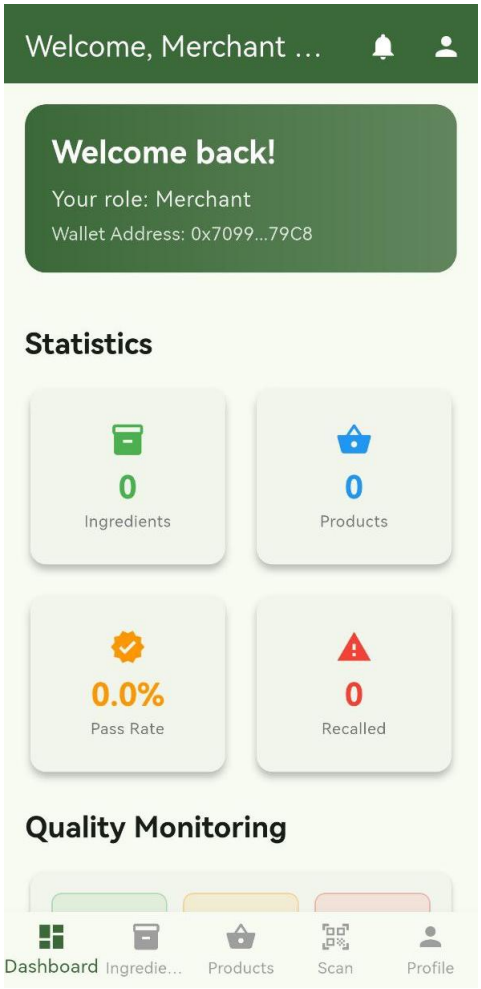


Figure 5.12 Merchant Dashboard (Part 1)

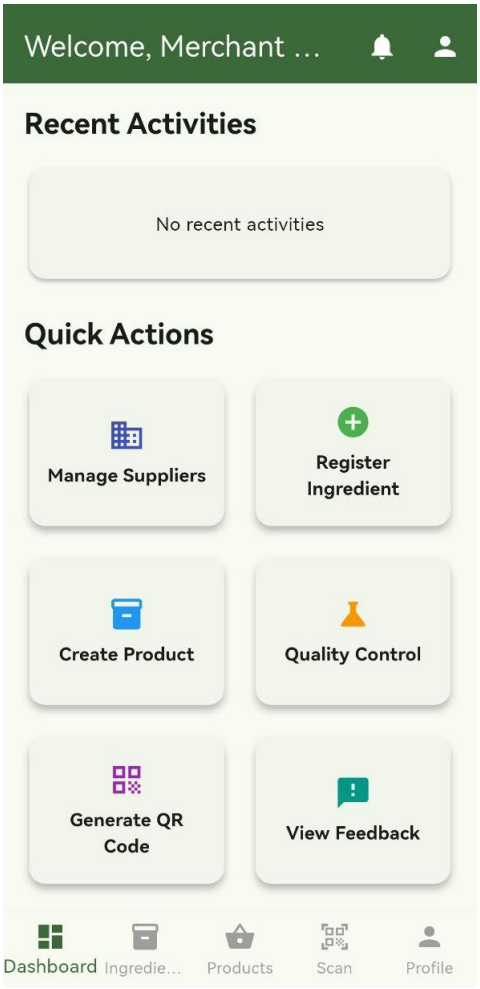


Figure 5.13 Merchant Dashboard (Part 2)

After a successful login, the merchant is navigated to the main dashboard (Figure 5.12 and 5.13), which serves as the central hub for system interaction. The interface is organized into four main sections:

1. Statistics: Provides an overview of the number of ingredients, products, and other key metrics related to the food supply chain.
2. Quality Monitoring: Displays the status of all recorded ingredients, allowing merchants to track storage conditions, contamination alerts, and overall quality compliance.
3. Recent Activities: Summarizes the latest actions performed by the merchant, such as ingredient registration, product creation, or recall events, offering quick visibility into ongoing processes.

4. Quick Actions: A shortcut panel that enables merchants to efficiently manage suppliers, register new ingredients, create products, quality control, generate QR codes for consumer verification, and view customer feedback.

This structured interface design ensures that merchants can seamlessly manage their operations while maintaining transparency and compliance within the Nutriguard system.

5.4.3 Supplier Registratrion

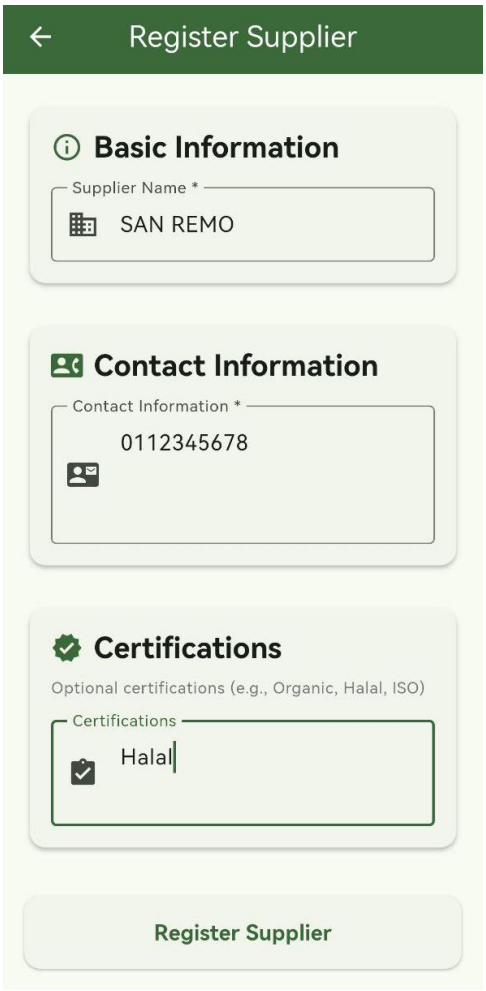


Figure 5.14 Register Supplier Form

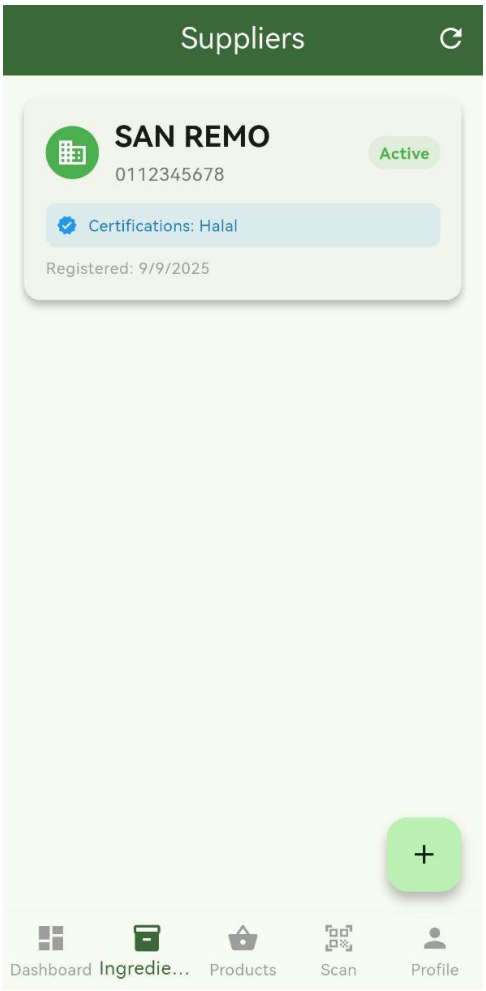


Figure 5.15 Suppliers list

First, the merchant needs to navigate to the Supplier Management page, where supplier information can be registered. Figure 5.14 shows the required fields include the supplier's name, contact details, and certifications (such as Halal, ISO, or Organic). After entering the details, the merchant can click the "Register Supplier" button to successfully complete the registration process. All submitted data will be uploaded to the blockchain, ensuring immutability and transparency. Additionally, merchants can

view the list of registered suppliers within the Supplier Management page (Figure 5.15), providing a clear overview of verified suppliers for future reference.

5.4.4 Ingredient Registration

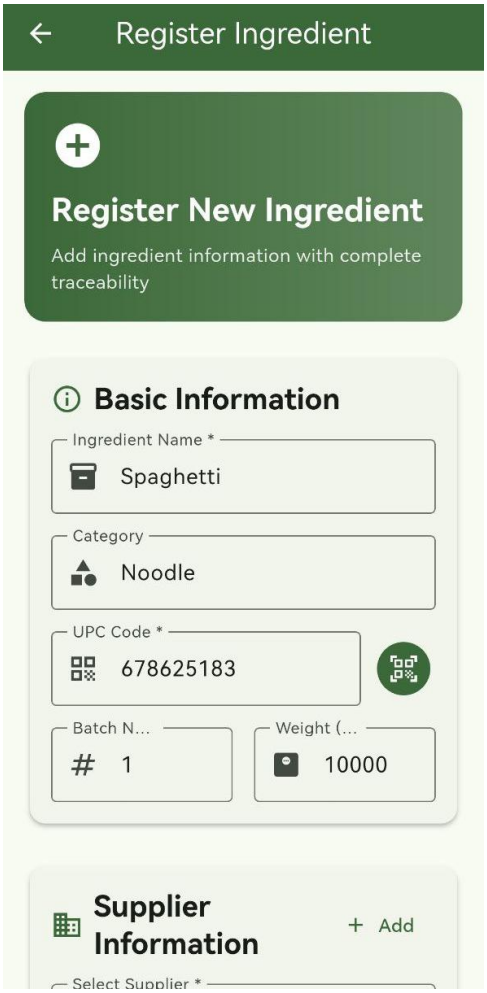


Figure 5.16 Register Ingredient Form

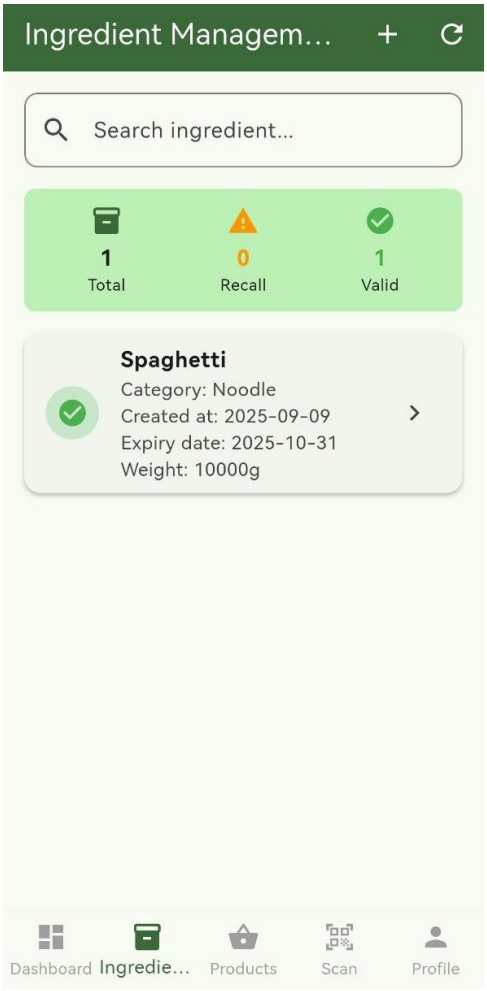


Figure 5.17 Ingredient list

Next, the merchant needs to access the ingredient management page and open the ingredient creation form to add a new ingredient. The required information includes the ingredient name, category, and UPC code which can either be entered manually or scanned using the camera, as well as the batch number and weight. The merchant must also select the registered supplier associated with the ingredient and provide additional details such as the production date, expiration date, and storage conditions as temperature and humidity. Once all the details are completed, clicking the “Register Ingredient” button will finalize the process, and the data will be securely uploaded to the blockchain. Merchants can then view the complete list of ingredients and their status within the ingredient management page.

5.4.5 Product Creation

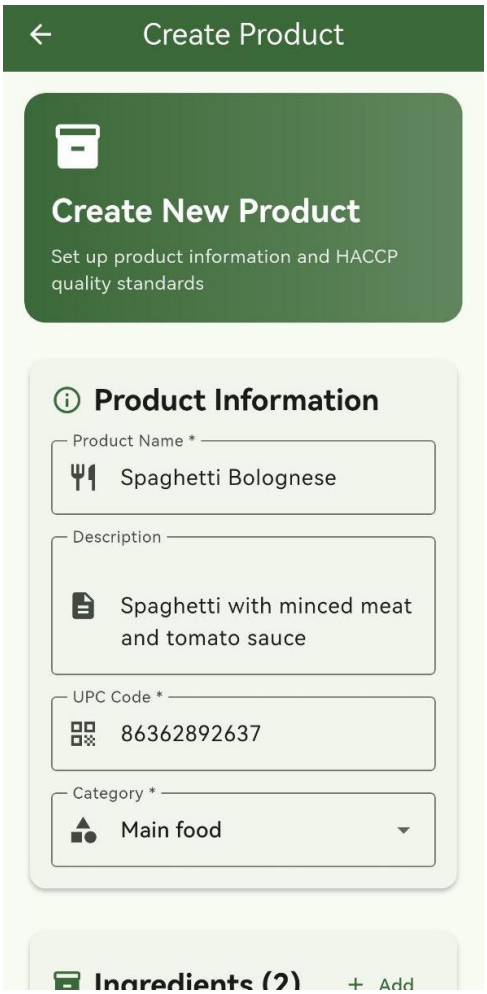


Figure 5.18 Create Product Form

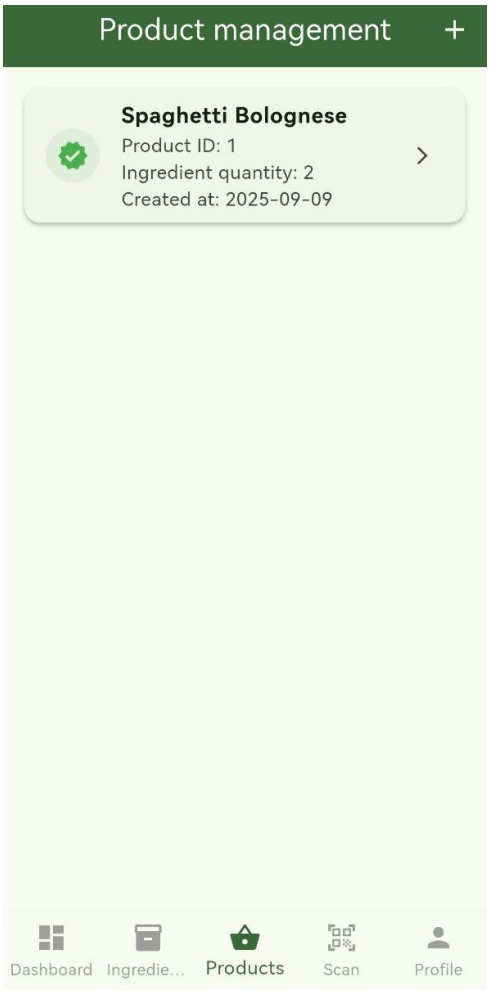


Figure 5.19 Product list

After registering the required ingredients, the merchant proceeds to the product management page to create a new product. Within the product creation form, the merchant is required to provide essential details such as the product name, description, UPC code, and category. Next, the merchant selects all the registered ingredients that are used in the product formulation. In addition, specific quality standards must be defined for the product, such as cooking temperature, humidity level, or pH value (in the case of beverages). Once all fields are completed, clicking the “Create Product” button uploads the data to the blockchain, thereby finalizing the registration process. The merchant can then view the complete list of products along with their status on the Product Management page.

5.4.6 Quality Control with IoT Reading

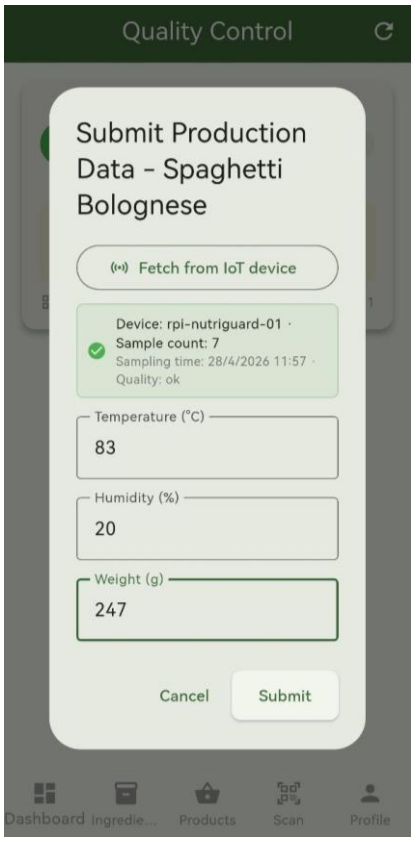


Figure 5.20 Fetch IoT Data

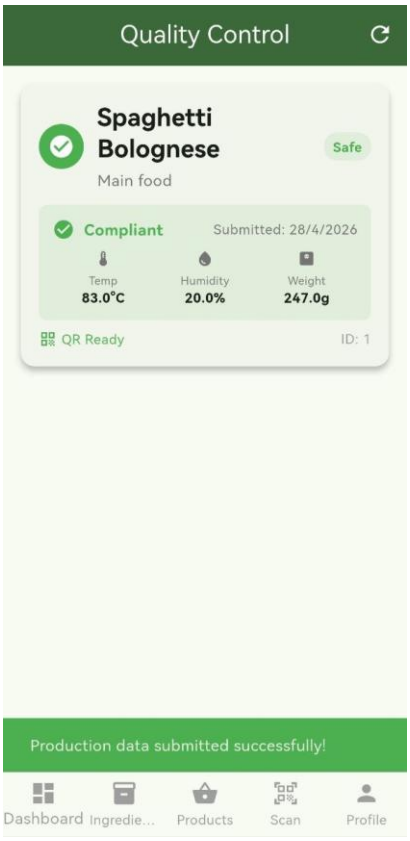


Figure 5.21 Pass Quality Control

The quality-control page lists merchant products. If a product has no production data, the merchant can submit production readings. The "Read from IoT Device" button calls the IoT gateway through IoTSensorService. Valid temperature and humidity values are auto-filled. The merchant manually enters weight and pH if applicable. If all checks pass, the product becomes safe and QR generation is enabled. Otherwise, the product status becomes alert and QR generation remains blocked.

5.4.7 QR Code Generation

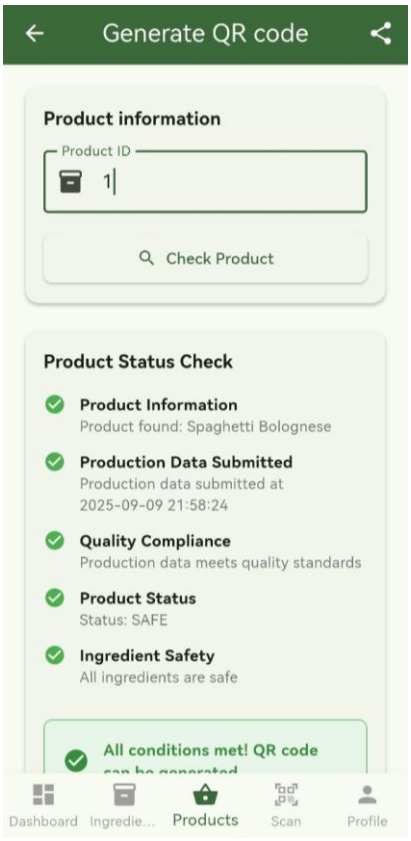


Figure 5.22 Pass the checking

Figure 5.23 Product QR Code

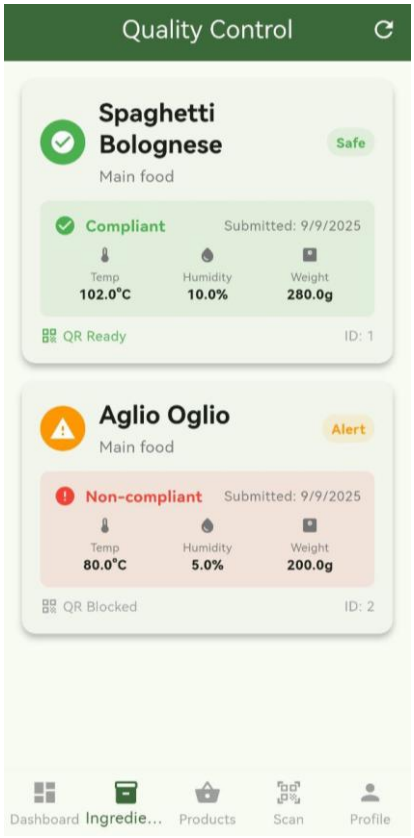


Figure 5.24 Fail the checking

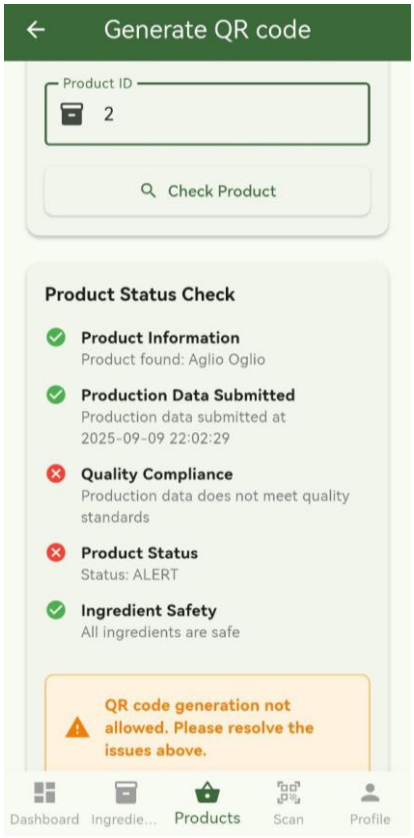


Figure 5.25 Fail generate QR Code

Subsequently, the merchant navigates to the QR Code Generation page and enters the product ID to generate the corresponding QR code. The generated QR serves as a unique identifier, enabling consumers to verify the product details and quality compliance. Figures above illustrate two cases: one where the product successfully passes the quality check, and another where the product fails to meet the required standards.

5.4.8 Consumer Login

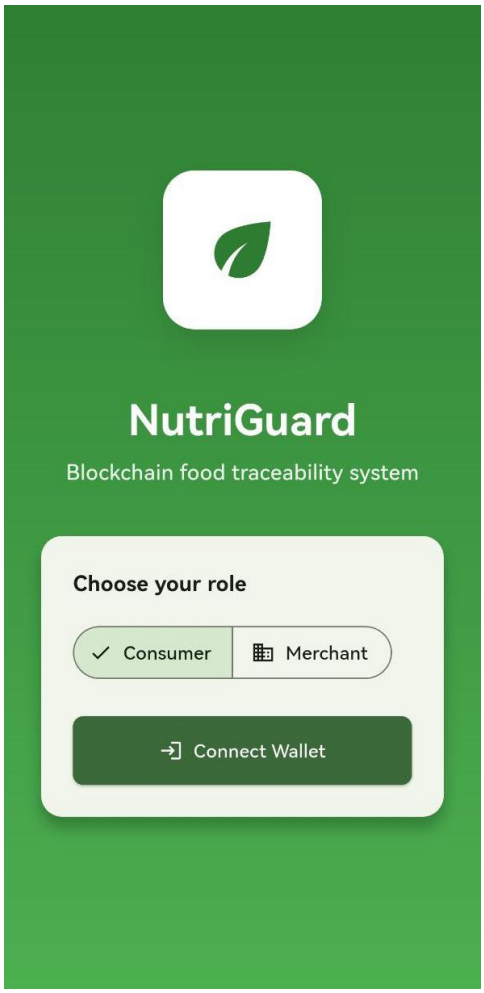


Figure 5.26 Consumer Login Page

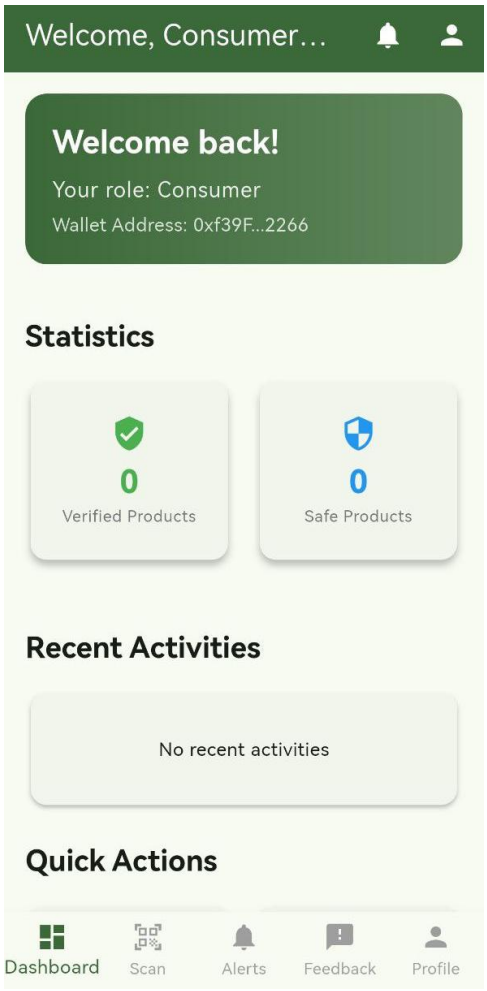


Figure 5.27 Consumer Dashboard

Figure 4.15 presents the consumer login page. After selecting the Consumer role and clicking the “Connect Wallet” button, the system navigates the user to the consumer dashboard as shown in Figure 4.16. The dashboard provides an overview of product-related information, including product statistics, recent activities, and a quick action menu for accessing key features.

5.4.9 Email Address Setting

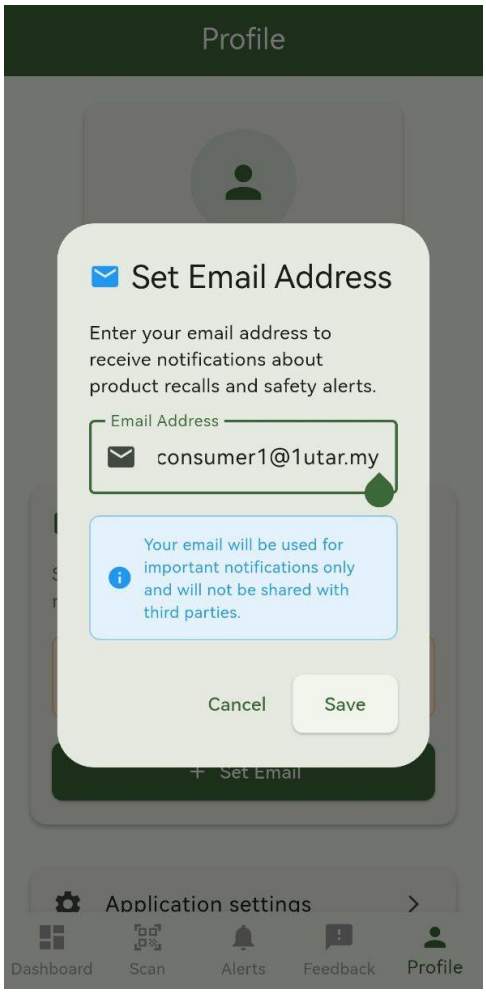


Figure 5.28 Insert email address

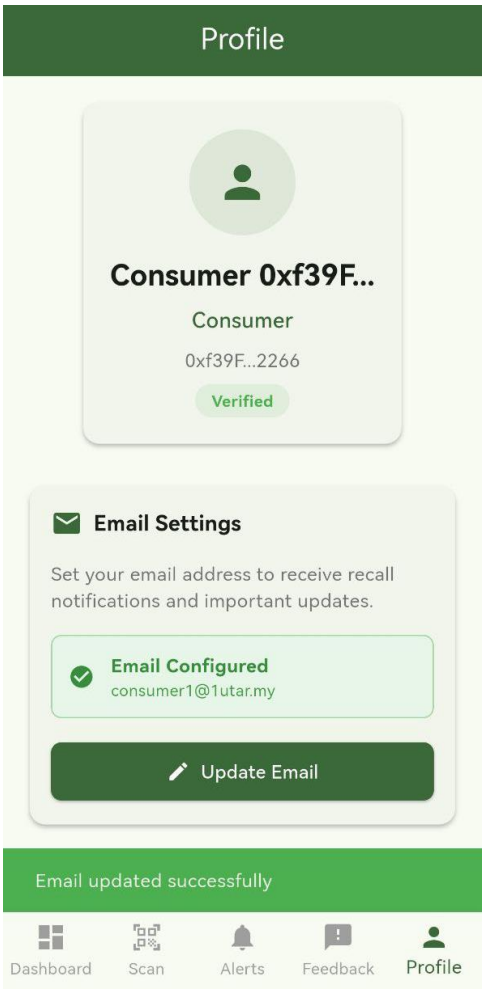


Figure 5.29 Successfully add email

Before scanning any product, the consumer is required to navigate to the Profile page to bind their email address. This step is essential for enabling the notification feature, where the system automatically sends an email alert if any registered product is identified as contaminated and subject to recall.

5.4.10 Consumer QR Scanning

Figure 5.30 Scanning page

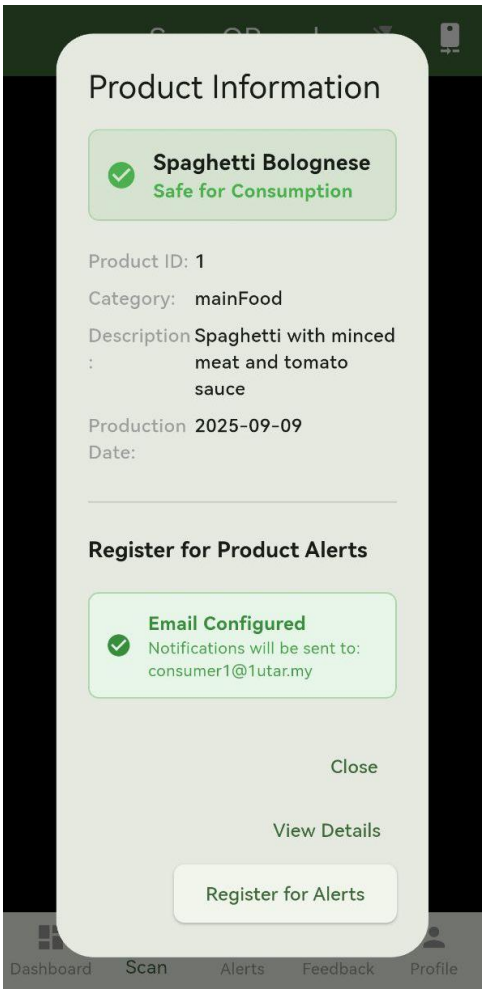


Figure 5.31 Successfully Scan the QR

After completing the profile setup, the consumer can proceed to the QR Scanning page to scan the QR code attached to the purchased product. Once the QR code is successfully scanned, the system displays the product's basic information, including its name, description, and associated batch details. At this stage, the consumer has the option to bind the product to their account. If this option is selected, any future recall notifications related to the product will automatically trigger an alert sent to the consumer's registered account.

5.4.11 Consumer Product Registration and Verification

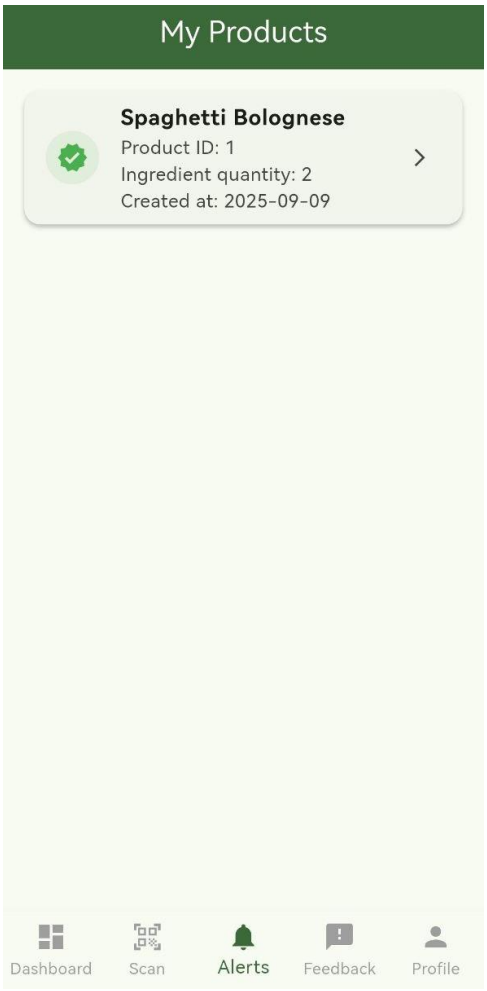


Figure 5.32 Registered product list

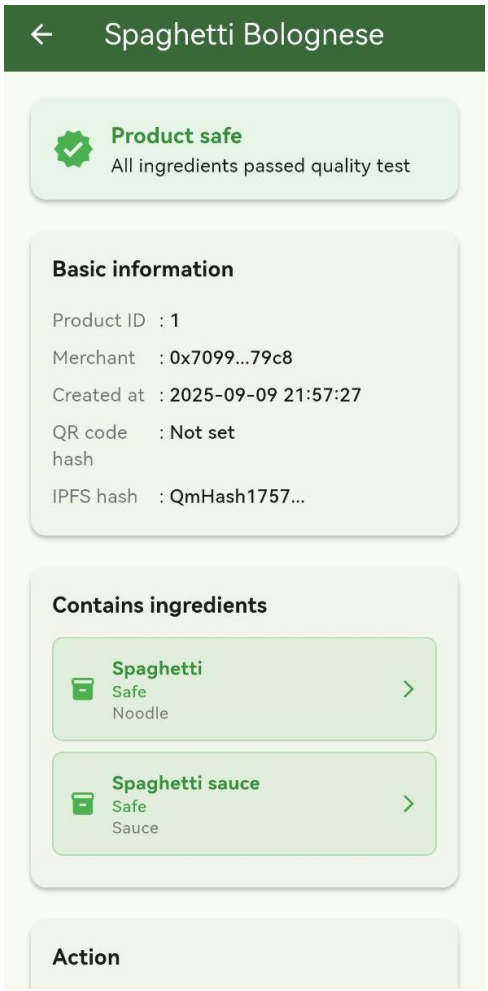


Figure 5.33 Product details

After successfully binding a product, the consumer can access the My Products page to view all products linked to their account. From this page, the consumer can select a specific product to review more detailed information, including the list of ingredients used, supplier certifications, and other relevant traceability records. This feature enhances transparency and allows consumers to verify the authenticity and quality compliance of the products they have purchased.

5.4.12 Consumer Product Rating and Feedback

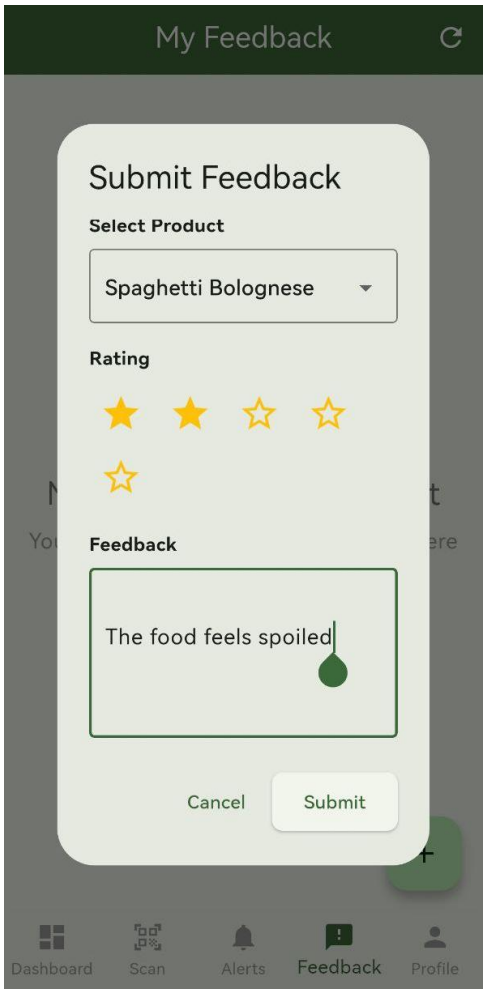


Figure 5.34 Feedback Form

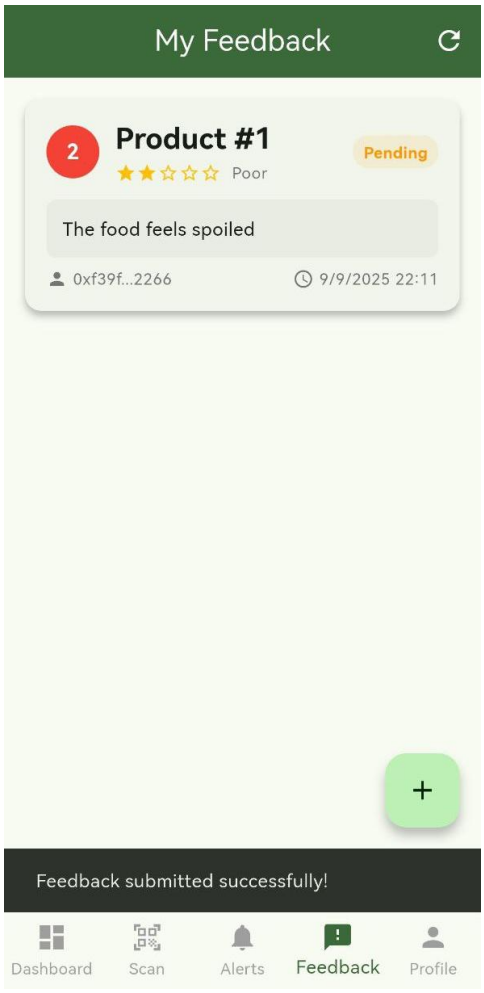


Figure 5.35 Feedback list

If consumers are dissatisfied with a product or suspect of potential issues, they can submit feedback directly through the application. The feedback will be forwarded to the merchant for further investigation, ensuring that concerns are properly documented and addressed. This feature not only empowers consumers to voice their concerns but also strengthens the accountability and responsiveness of merchants in maintaining food quality and safety.

5.4.13 Feedback Processing

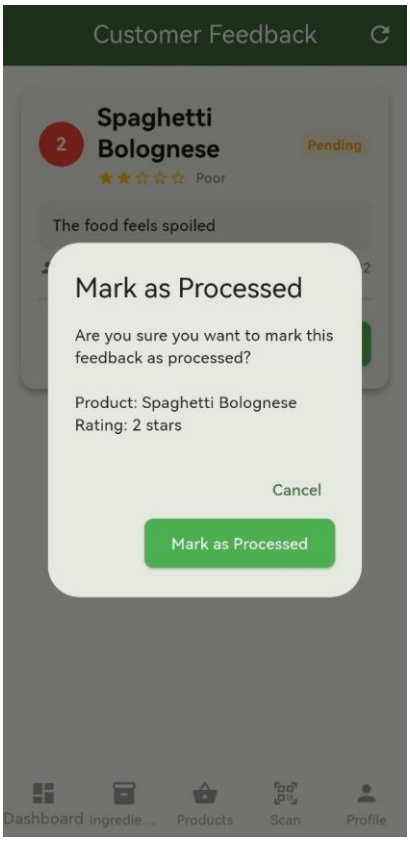


Figure 5.36 Mark Feedback

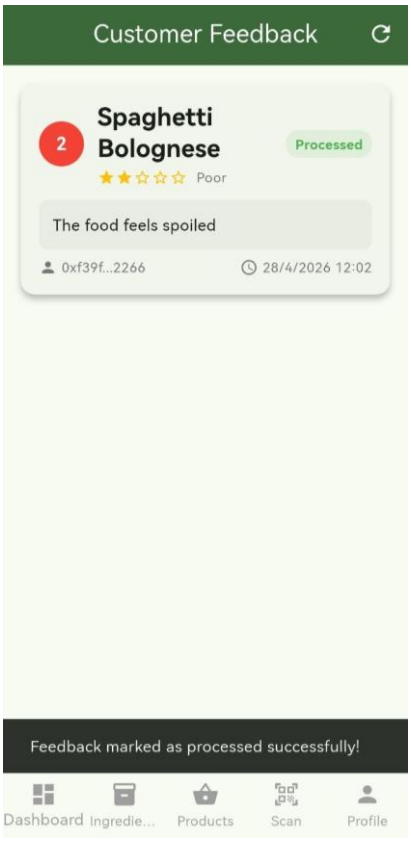


Figure 5.37 Feedback Processed

After consumers submit feedback, merchants can review all feedback records associated with their own products. Each feedback entry includes product ID, consumer address, rating score, feedback text, submission time, and processing status.

The merchant-side processing flow is:

1. Open feedback list page.
2. Filter or select feedback by product.
3. Review content and verify whether follow-up action is needed.
4. Mark feedback as processed.

When the merchant confirms processing, the app calls markFeedbackAsProcessed(feedbackId). The smart contract enforces ownership checks so only the product owner can process feedback. A FeedbackProcessed event is emitted and the UI is refreshed to show the updated status.

5.4.14 Initiate Recall

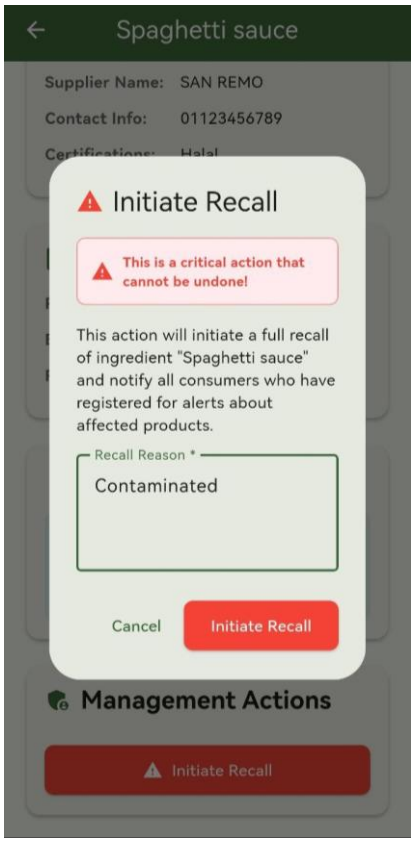


Figure 5.36 Initiate Recall and enter Reason

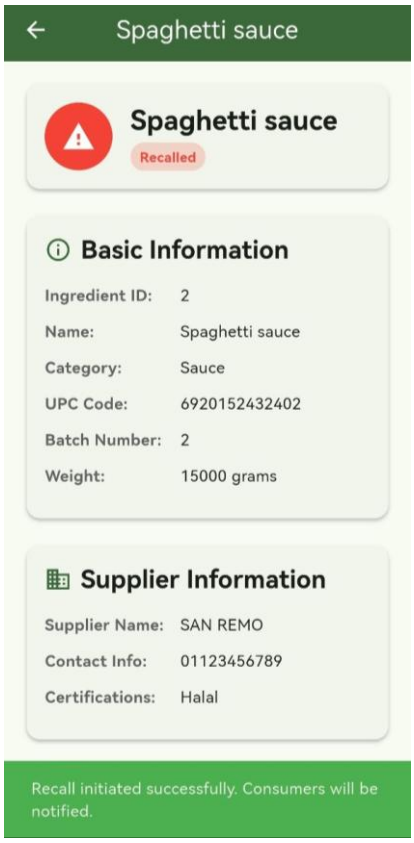


Figure 5.37 Recall Successfully

When contamination is detected, the merchant can initiate recall from the ingredient management or product traceability flow. The merchant enters a recall reason and confirms submission. After transaction confirmation, the contract performs cascade updates through _updateAffectedProductsStatus(), which:

1. Marks the selected ingredient as recalled/contaminated.
2. Iterates through products that reference the ingredient.
3. Sets affected products to Contaminated.
4. Sets isValid = false and canGenerateQR = false.
5. Emits RecallInitiated event for auditability.

This ensures unsafe products are blocked from QR usage immediately after recall confirmation.

5.4.15 Consumer Receive Alert

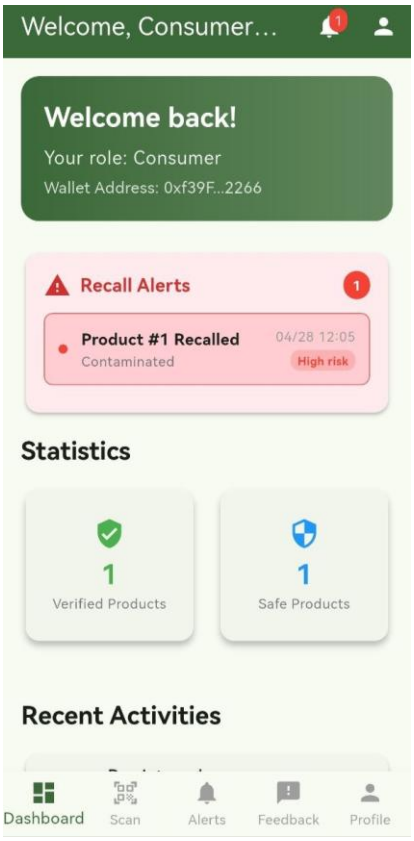


Figure 5.38 Alert in Dashboard

Figure 5.39 Alert Details

In the current prototype, consumers can register product alerts on-chain using registerForProductAlerts function. This stores consumer-product registration data and supports traceable recall targeting. When a recall is initiated, consumers who scan or view the product can see updated contaminated/recall status from on-chain state. After that, registered consumer records can be retrieved for off-chain notification processing. The contract-level registration acts as the source of truth for downstream alert services.

5.5 System Implementation Issues and Challenges

5.5.1 Sensor Accuracy and Reliability

DHT22 has limited accuracy and can produce occasional read errors due to timing sensitivity. The gateway addresses this by ignoring failed readings and averaging recent valid samples through a sliding window. For production use, SHT31, or industrial food-grade sensors should be considered.

5.5.2 Local Network Dependency

The Flutter app, Raspberry Pi, and Hardhat node must be reachable on the same local network during demonstration. Mobile network isolation, firewall rules, and cleartext HTTP restrictions can cause connection errors.

5.5.3 Blockchain Deployment Scope

The current prototype is demonstrated on a Hardhat local blockchain. This is suitable for rapid testing but does not fully represent public network transaction fees, confirmation delay, and wallet UX. Sepolia or Layer-2 testnet deployment is recommended.

5.5.4 Consumer Email Privacy

The prototype stores email strings in smart contract state. This is not recommended for production because blockchain data is public and immutable. A production design should store encrypted off-chain contact details and only place hashes or references on-chain.

Chapter 6

SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

The evaluation of NutriGuard is conducted to assess both functional correctness and system behavior under a realistic blockchain-enabled food traceability scenario. Unlike conventional application testing, this system integrates decentralized smart contracts, IoT-based data acquisition, and mobile interaction, requiring a multi-dimensional evaluation approach.

The evaluation is structured across four key dimensions:

- Functional correctness: Whether system features operate as intended
- IoT reliability: Whether sensor data is consistently available and valid
- Smart contract behavior: Whether on-chain logic enforces rules correctly
- End-to-end workflow validation- Whether the complete supply chain scenario functions correctly

The main evaluation metrics are:

1. Functional success rate: Whether each feature completes successfully under expected conditions.
2. Validation correctness: Whether production data correctly changes product status to safe or alert.
3. Recall correctness: Whether recalled ingredients correctly invalidate affected products.
4. IoT gateway response: Whether */sensor* and */latest* API endpoint returns valid readings when the sensor is connected and stale when readings are unavailable.
5. Transaction execution: Whether smart contract transactions are mined successfully on the Hardhat network.
6. User workflow completion: Whether merchants and consumers can complete the intended workflows without system crashes.

The evaluation is designed to validate not only functional correctness, but also system behavior under realistic blockchain and IoT integration conditions.

6.2 Testing Setup and Result

6.2.1 Testing Environment

The testing environment simulates a local blockchain-enabled supply chain system where all components (mobile app, IoT gateway, and blockchain node) operate within the same network to ensure controlled and repeatable evaluation.

Item	**Specification / Configuration**
Development OS	Windows 11
Blockchain Network	Ethereum Hardhat local network, chain ID 1337
Smart Contract	NutriGuard.sol, Solidity 0.8.28
Smart Contract Address	0xC1707AAa3b0dc69438c0122E4985586A811011c1
Hardhat Local Network Configuration	http://127.0.0.1:8545
Currency Symbol	ETH
Wallet	Metamask
Mobile App	Flutter application running on Android device
IoT Gateway	Raspberry Pi 3 model B+
Sensor	DHT22 temperature and humidity sensor

Table 6.1 Specification and Configuration of Testing environment

6.2.2 Metamask Setup

MetaMask is used as the blockchain wallet for user authentication and transaction signing within the NutriGuard system. It enables secure interaction between the Flutter application and the local Hardhat blockchain network.

All blockchain interactions in NutriGuard require user authorization via MetaMask. When a user performs actions, MetaMask prompts the user to approve and sign the transaction. Once approved, the transaction is sent to the blockchain and recorded immutably

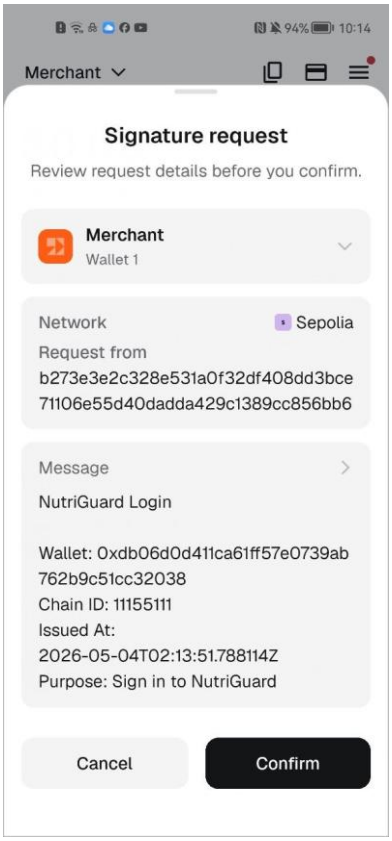
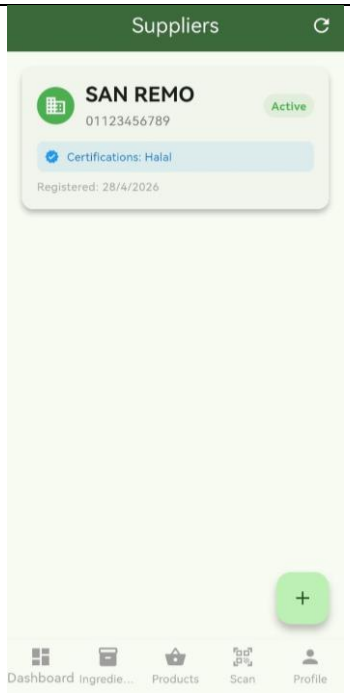
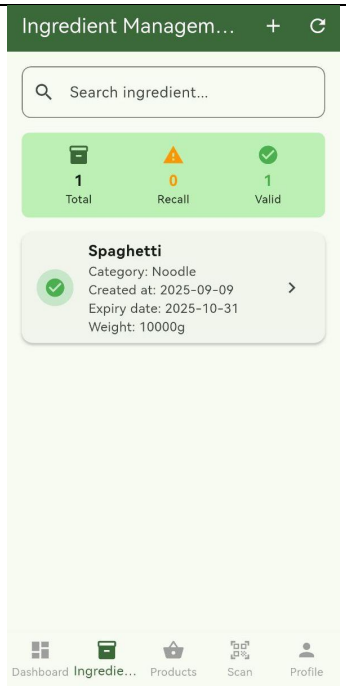
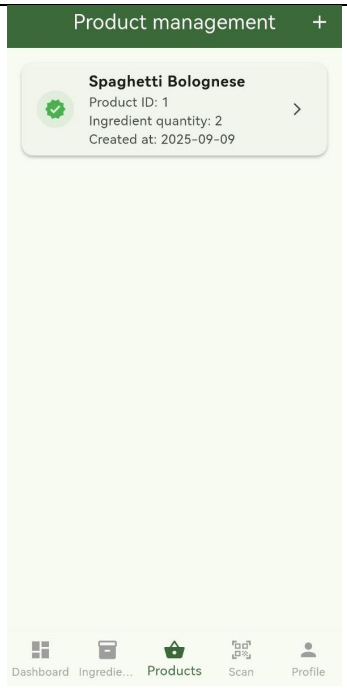
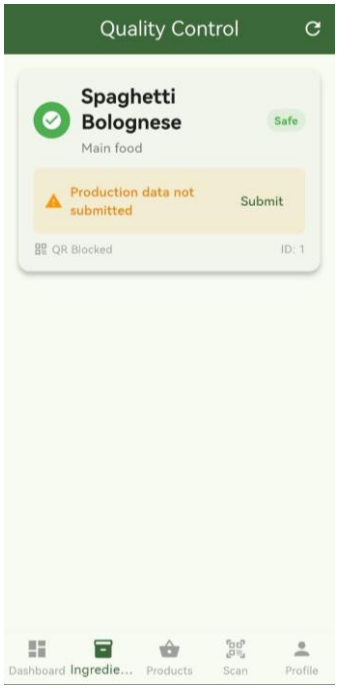


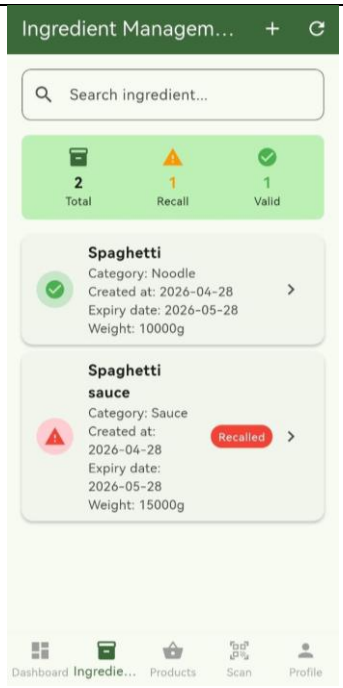
Figure 6.1 Transaction Confirmation

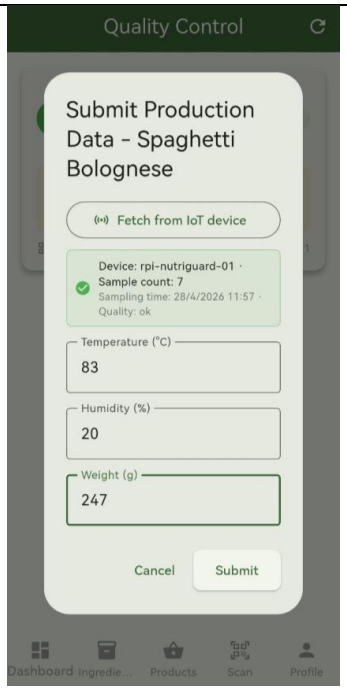
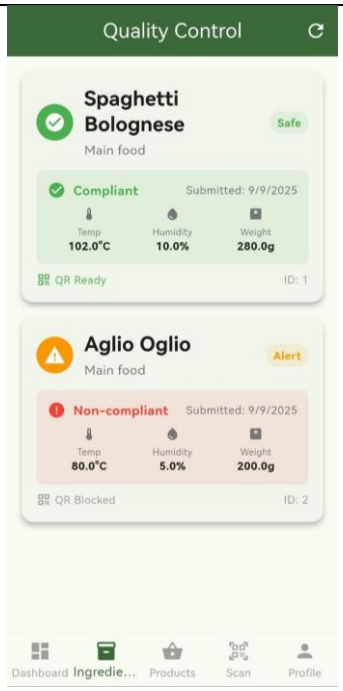
6.2.3 Functional Test Cases

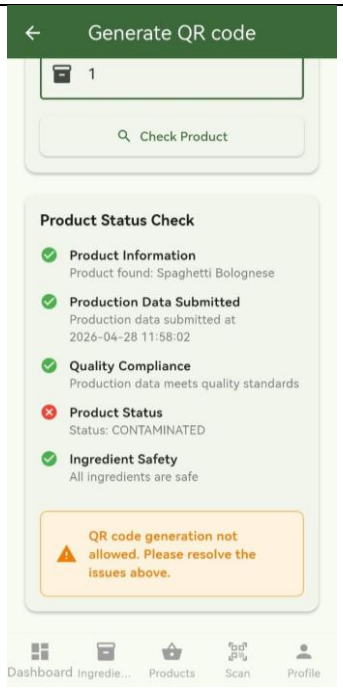
The functional tests cover all critical system operations including data registration, validation, recall, and user interaction.

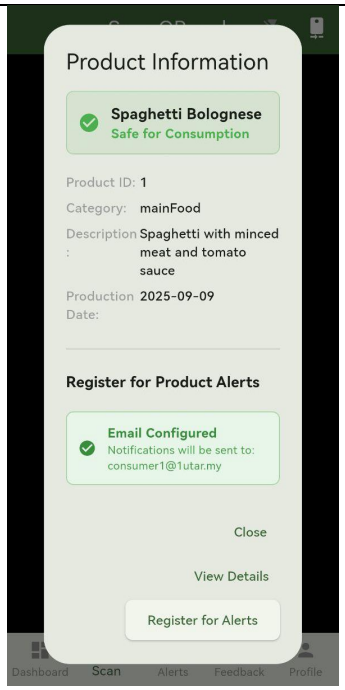
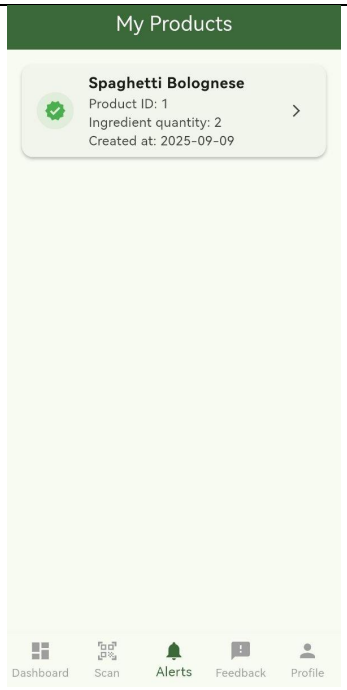
Test ID	Test Case	Expected Result	Actual Result	Status
FT01	Register supplier	Supplier stored on-chain and appears in supplier list		Pass
FT02	Register ingredient	Ingredient stored on-chain and linked to merchant		Pass

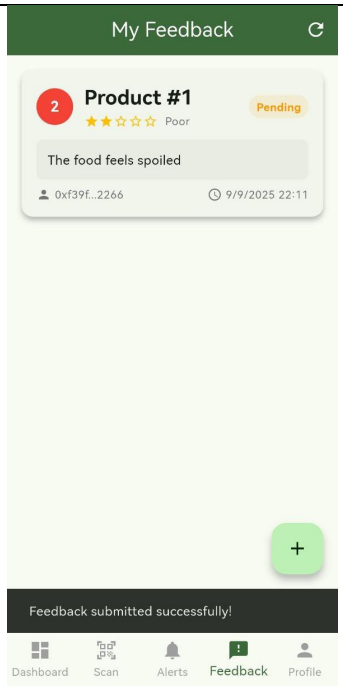
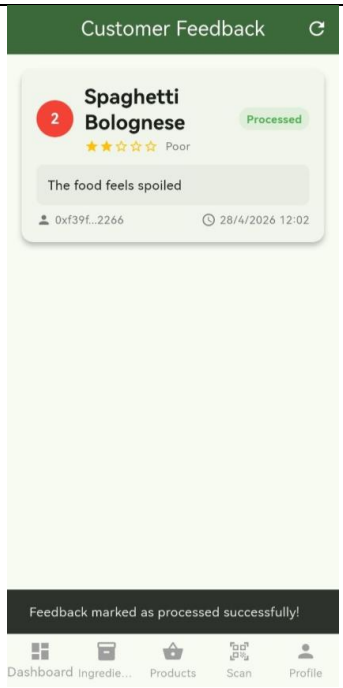
FT03	Create product with valid ingredients	Product created and initially cannot generate QR	 	Pass

FT04	Create product with recalled ingredient	Contaminated ingredient not display in list	 	Pass

FT06	Fetch IoT reading from gateway	Temperature and humidity auto-fill form		Pass
FT07	Submit non-compliant production data	Product status becomes Alert and QR blocked		Pass

FT08	Generate QR for safe product	QR code generated successfully		Pass
FT09	Generate QR for alert product	QR code will not display		Pass

FT10	Scan QR as consumer	Product details displayed		Pass
FT11	Register product alert	Consumer registration stored on-chain		Pass

FT12	Submit consumer feedback	Feedback stored and visible to merchant		Pass
FT13	Process feedback	Feedback status updated to processed		Pass

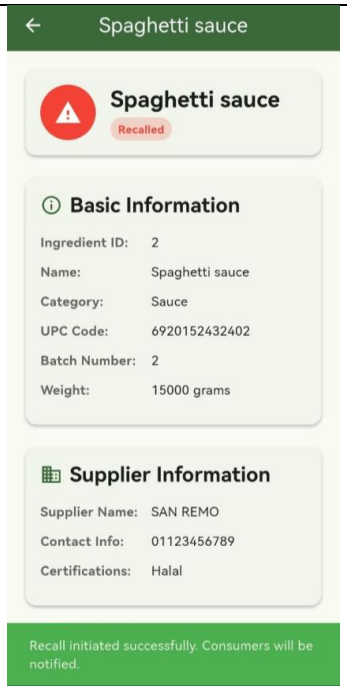
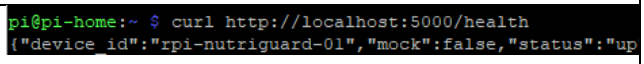
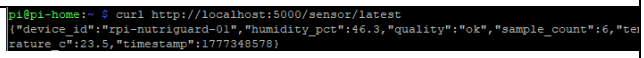
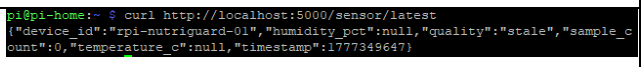
FT14	Initiate ingredient recall	Ingredient recalled and affected products contaminated		Pass

Table 6.2 Functional Test Case of Integration Testing

All functional tests passed successfully, indicating that the system correctly enforces business rules such as preventing recalled ingredient usage, blocking QR generation for unsafe products and maintaining correct state transitions

6.2.4 IoT Edge Gateway Test Results

Test ID	Scenario	Expected Result	Actual Result
IT01	/health endpoint	Returns status up and device ID	
IT02	Sensor connected	/sensor/latest returns quality=ok	
IT03	Sensor disconnected or no	/sensor/latest returns	

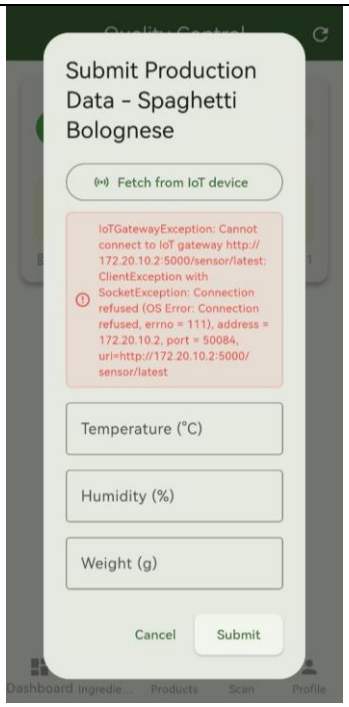
	fresh sample	quality=stale	
IT04	Flutter gateway unavailable	App shows error without crash	

Table 6.3 IoT Gateway Testing

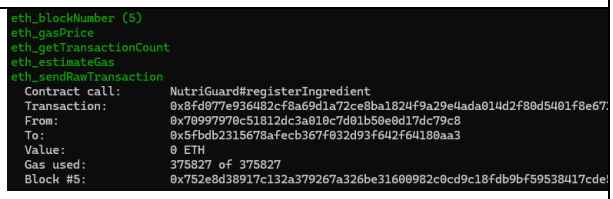
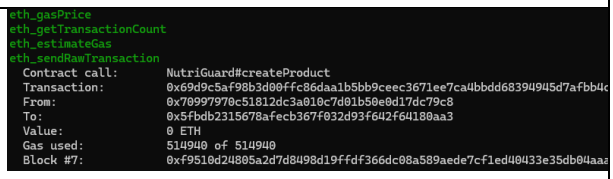
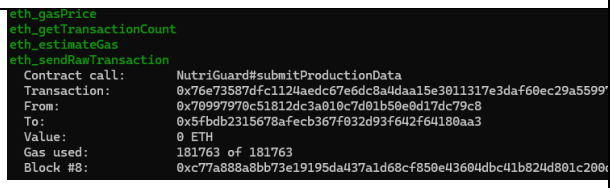
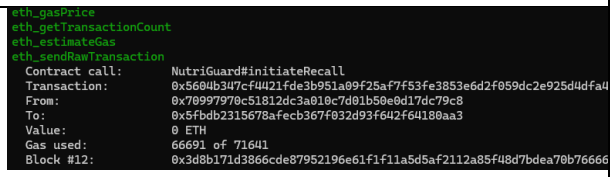
The IoT gateway was evaluated in terms of reliability, response behavior, and failure handling to determine its effectiveness in supporting production quality validation. Under normal operating conditions, when the sensor is properly connected, the gateway consistently returns valid temperature and humidity readings with a quality status of "ok". This indicates that the system can reliably acquire environmental data for automated quality control.

In failure scenarios, such as when the sensor is disconnected or when no recent readings are available, the system correctly returns a quality status of "stale". This prevents outdated or invalid data from being used in production validation. Additionally, when the gateway is completely unavailable, the mobile application displays an error message while maintaining system stability without crashing. This demonstrates robust error handling and system resilience.

Despite these strengths, several limitations are identified. The DHT22 sensor used in this prototype has limited accuracy and may produce fluctuating readings under certain conditions. Furthermore, the system relies on local network connectivity, which may affect data availability in real-world deployment scenarios. Overall, the IoT gateway

enhances automation but cannot be fully trusted, highlighting the importance of validation mechanisms and manual fallback options.

6.2.5 Smart Contract Transaction Evaluation

Function	Expected Behavior	Gas Used and Result
registerSupplier	Stores supplier record	
registerIngredient	Stores ingredient record	
createProduct	Stores product and quality rule	
submitProductionData	Validates data and updates status	
initiateRecall	Updates ingredient and affected products	

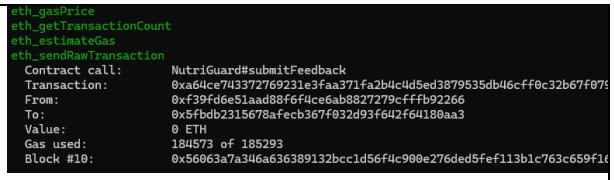
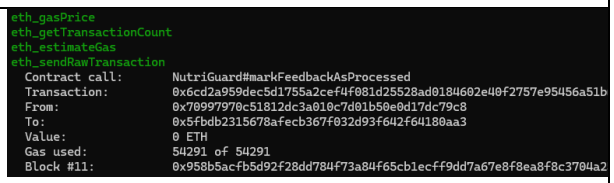
submitFeedback	Stores consumer feedback	
markFeedbackAsProcessed	Update feedback status as processed	

Table 6.4 Functional Test for Smart Contract

The smart contract was evaluated based on correctness, access control, and execution behavior to ensure that it reliably enforces system rules and maintains data integrity. The results show that the validation logic operates correctly, as production data submissions accurately update product status to either safe or alert based on predefined quality thresholds. In addition, the recall cascade mechanism functions as intended, automatically identifying and updating all products associated with a contaminated ingredient.

Access control is effectively enforced through require conditions embedded in the smart contract. Only authorized merchants are permitted to create, modify, or update supply chain data, while unauthorized or invalid operations are rejected. This ensures that system integrity is preserved and prevents misuse of blockchain functions.

From an execution perspective, all transactions were successfully mined on the Hardhat local blockchain, demonstrating consistent and deterministic behavior. The contract execution remains predictable across repeated tests, confirming its reliability. Although gas consumption is not a primary concern in the local testing environment, it is observed that operations such as recall propagation may incur higher costs when a large number of products are affected.

Overall, the smart contract demonstrates strong correctness and enforcement capabilities, ensuring that business rules are applied consistently while preventing unsafe operations within the system.

6.2.6 IPFS Test

The IPFS component of NutriGuard was evaluated to verify its role in supporting off-chain data storage through hash referencing. Unlike on-chain data, which is stored directly on the blockchain, IPFS is used to store ingredient and product-related documents such as images or certificates, with only the resulting content hash recorded in the smart contract.

During testing, selected product-related files were manually uploaded to the IPFS network via a pinning service. Upon successful upload, a CID was generated. This CID was then stored in the smart contract as part of the ingredient or product record.

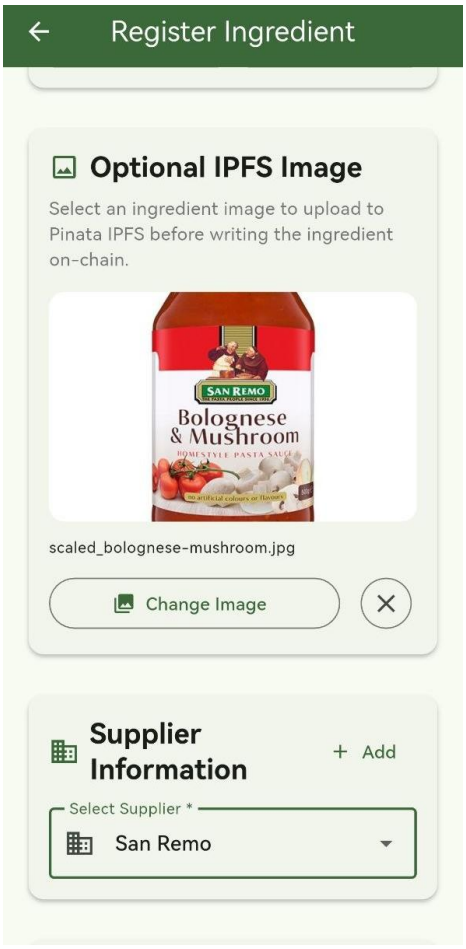


Figure 6.2 Image of Ingredients

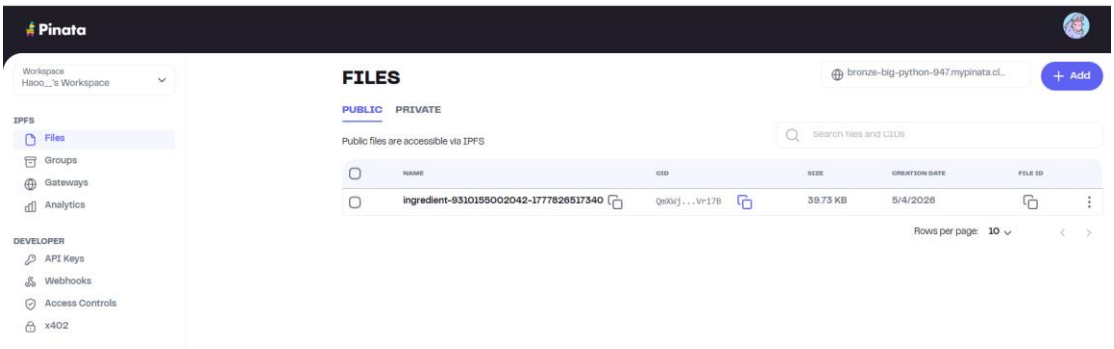


Figure 6.3 Pinata Dashboard

To validate the integration, stored IPFS hashes were retrieved from the blockchain and used to access the corresponding files. The retrieved files matched the originally uploaded content, demonstrating that the system correctly links on-chain records with off-chain data. This mechanism enables users to verify that a file has not been altered, as any change would produce a different hash.

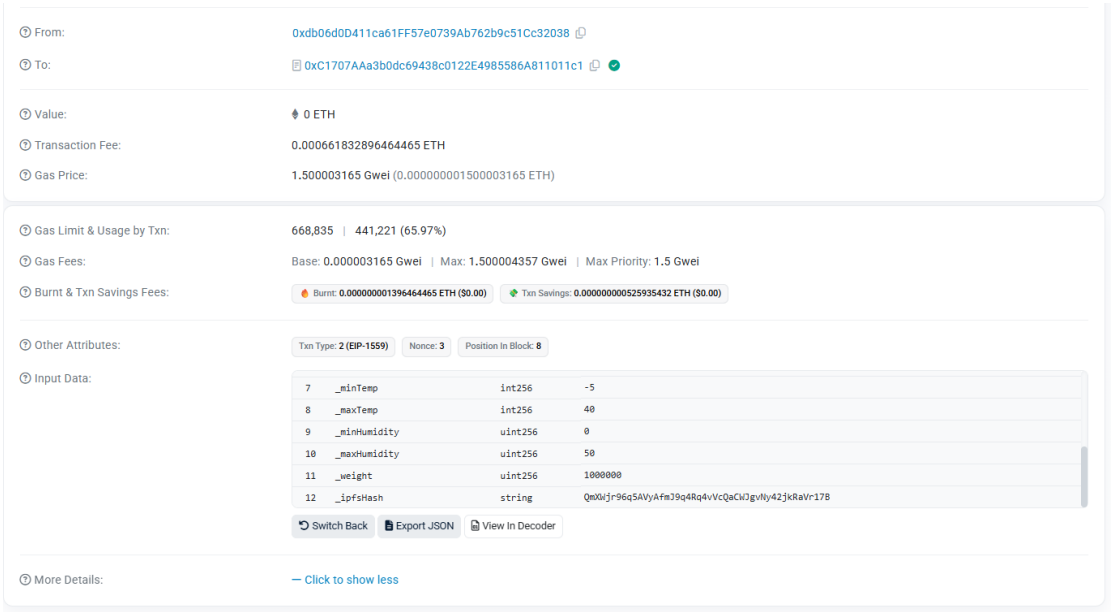


Figure 6.4 Transaction Details

6.2.7 End-to-End Scenario Test

An end-to-end scenario should include:

1. Merchant logs in.
2. Merchant registers supplier.
3. Merchant registers ingredient.
4. Merchant creates product with HACCP thresholds.
5. Merchant fetches IoT temperature and humidity from Raspberry Pi.

6. Merchant submits production data.
7. Smart contract marks product safe or alert.
8. Merchant generates QR code if product is safe.
9. Consumer scans QR code.
10. Consumer registers product alert and submits feedback.
11. Merchant initiates ingredient recall.
12. Affected product becomes contaminated.
13. Consumer sees recalled/contaminated status.

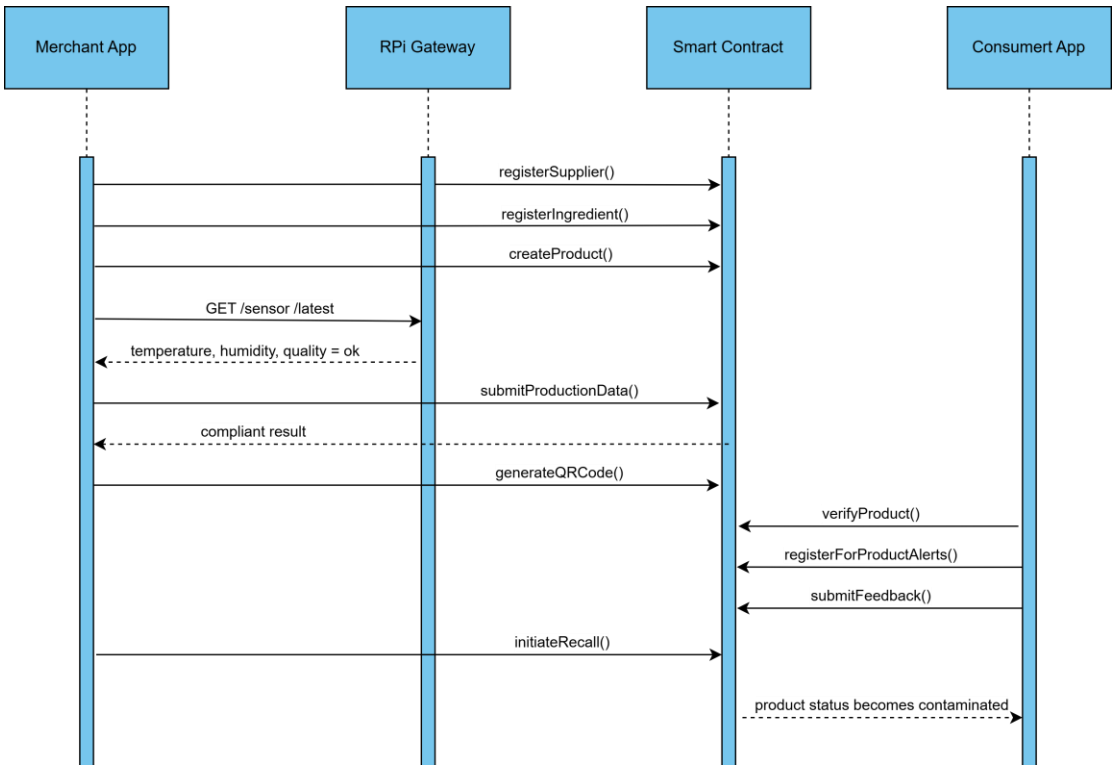


Figure 6.5 End-to-end Testing

6.3 Project Challenges

Several challenges were encountered during the development and evaluation of the NutriGuard system, particularly due to the integration of blockchain, IoT, and mobile technologies.

One major challenge was synchronizing blockchain state with the mobile user interface. Since blockchain transactions are asynchronous and may experience delays due to network conditions or validation processes, the application must handle delayed

responses and ensure that updated data is accurately reflected in the UI. This was addressed by implementing refresh mechanisms and clear user feedback messages.

Another challenge involved IoT gateway reliability. Sensor readings from the DHT22 device were occasionally inconsistent or unavailable, which could impact the accuracy of production validation. To mitigate this, the system incorporates stale-data detection and allows manual data input as a fallback mechanism when sensor data is unreliable. Network configuration also posed a challenge, as the mobile device, Raspberry Pi gateway, and Hardhat blockchain node must operate within the same network. Incorrect IP configuration could prevent communication between components. This issue was resolved by ensuring consistent network setup and enabling external access to the local blockchain node.

In addition, the smart contract design evolved throughout the development process, transitioning from an earlier model to the final merchant-consumer structure. This required updates to previously defined test cases and system logic. Finally, privacy concerns were identified due to the storage of consumer email addresses on-chain. While acceptable for a prototype, this approach is not suitable for production environments and highlights the need for secure off-chain storage in future implementations.

6.4 Objective Evaluation

Objective	Achievement Level	Evidence / Justification
Develop blockchain traceability across supplier, ingredient, product, production, QR, recall, and feedback data	Achieved	All entities are stored and managed through the NutriGuard smart contract and validated through functional test cases
Integrate IoT devices for real-time monitoring	Achieved	Temperature and humidity are automatically retrieved from the Raspberry Pi gateway
Provide merchants with automated quality control and recall management	Achieved	Smart contract enforces quality validation rules and successfully executes recall cascade to update affected products
Provide consumers with QR traceability and feedback	Achieved	Consumers can scan QR codes to view product information, register for alerts, and submit feedback
Demonstrate feasibility in a restaurant-like encironment	Achieved	End-to-end scenario testing confirms that the system workflow operates correctly in a simulated environment

Table 6.5 Summary Table of Objective Evaluation

6.5 Discussion

The evaluation of NutriGuard reveals several important trade-offs, limitations, and opportunities for future improvement. One key trade-off lies between decentralization and system performance. While the use of blockchain ensures data immutability, transparency, and trust, it also introduces latency due to transaction confirmation and network interaction. This can affect system responsiveness, particularly when multiple transactions are required in sequence.

Another important design consideration is the balance between on-chain and off-chain storage. Storing data on-chain enhances integrity and auditability, as records cannot be modified once committed. However, this approach increases storage costs and reduces scalability. To address this, NutriGuard utilizes IPFS for off-chain storage of large or unstructured data, with only hash references stored on-chain. While this improves efficiency, it introduces dependency on external storage availability and does not fully guarantee data persistence without a complete upload and verification pipeline.

The integration of IoT devices introduces an additional trade-off between automation and reliability. Automated data collection from sensors such as the DHT22 improves efficiency and reduces manual input, but sensor readings may be inconsistent or inaccurate under certain conditions. As a result, the system incorporates validation mechanisms and manual fallback options to ensure that incorrect data does not compromise production quality assessment.

Despite its successful implementation as a prototype, the system has several limitations. First, the use of a Hardhat local blockchain environment does not accurately reflect real-world deployment conditions, such as network latency, gas cost variability, and node decentralization. Second, the current IPFS integration is limited to storing hash references, without a fully automated upload and verification pipeline. Third, the system does not include a real-time notification mechanism for consumers, as recall alerts must be manually checked through the application. Additionally, the DHT22 sensor used in the prototype has limited precision, which may affect the reliability of environmental monitoring. Finally, storing consumer email addresses directly on-chain raises privacy concerns and is not suitable for production environments.

To address these limitations, several future improvements are proposed. The integration of trusted oracle services can enhance the reliability of IoT data by securely bridging off-chain sensor inputs with on-chain validation logic. Sensitive user data, such as email addresses, should be stored off-chain using encryption or hashing techniques to improve privacy protection. Deploying the system on a public blockchain testnet, such as Sepolia, would allow evaluation under realistic network conditions and provide

insights into scalability and performance. Furthermore, implementing an automated IPFS upload and verification pipeline would strengthen data integrity and consistency. Finally, integrating real-time notification mechanisms, such as email or push notifications, would significantly improve user experience by enabling proactive recall alerts.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

This project designed and implemented NutriGuard, a blockchain-based food traceability and recall prototype. The system records supplier, ingredient, product, production, recall, consumer-registration, and feedback information through an Ethereum-compatible smart contract. It provides merchants with supplier management, ingredient management, product creation, HACCP-inspired quality-control validation, QR generation, recall initiation, and feedback processing. It provides consumers with QR scanning, product traceability, alert registration, and feedback submission.

The project also integrates IoT-assisted quality control through a Raspberry Pi 3B+ and DHT22 sensor. A Flask edge gateway collects temperature and humidity readings, applies sliding-window smoothing, and exposes REST endpoints for the Flutter application. This reduces reliance on manual entry for environmental parameters and demonstrates how physical sensor data can support blockchain-based production validation.

The most important recall-related contribution is the ingredient-level cascade mechanism. When an ingredient is marked as recalled or contaminated, all products using that ingredient are automatically updated to contaminated status and blocked from QR generation. This supports rapid recall and improves traceability accountability.

Although the prototype demonstrates feasibility, it is not yet production-ready. The current implementation runs mainly on a local Ethereum Hardhat blockchain, uses preset accounts for demonstration, stores email strings on-chain. These limitations provide clear directions for future enhancement.

7.2 Recommendation

Future work should focus on the following improvements:

1. Deploy to public testnet or Layer-2 testnet: Sepolia, Polygon, or Arbitrum test deployment should be performed to evaluate real transaction confirmation time, gas cost, and wallet interaction.
2. Build an off-chain notification gateway: Listen to recall events and send actual email or push notifications to registered consumers. Sensitive contact details should be stored off-chain and encrypted.
3. Upgrade IoT hardware: Add HX711 load-cell weight measurement and a pH sensor for beverage products. Consider replacing DHT22 with SHT31, or industrial-grade sensors.
4. Conduct usability evaluation: Use a System Usability Scale survey with at least 10-20 participants to measure consumer and merchant usability.
5. Improve privacy: Avoid storing raw emails on-chain. Use hashed identifiers, encrypted off-chain storage, or zero-knowledge approaches for privacy-preserving registration.
6. Improve scalability: Evaluate batching, event-only storage, and Layer-2 deployment to reduce gas cost and improve throughput.

REFERENCES

[1] J. G. Wen, X. J. Liu, Z. M. Wang, T. F. Li, and M. L. Wahlqvist, "Melamine-contaminated milk formula and its impact on children.," *PubMed*, vol. 25, no. 4, pp. 697–705, Dec. 2016

[2] H. F. Program, "Investigation Summary: Factors Potentially Contributing to the Contamination of Romaine Lettuce Implicated in the Fall 2018 Multi-State Outbreak of E. coli O157:H7," *U.S. Food And Drug Administration*, Feb. 13, 2019.

[3] D. Rodeck, "Understanding blockchain technology," *Forbes Advisor*, Mar. 04, 2025.

[4] V. Buterin, "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM."

[5] Y. Lu, "The blockchain: State-of-the-art and research challenges," *Journal of Industrial Information Integration*, vol. 15, pp. 80–90, Apr. 2019

[6] B. Academy, "Smart contract," Binance Academy.

[7] X. Wu and Y. Lin, "Blockchain recall management in pharmaceutical industry," Procedia CIRP, vol. 83, pp. 590–595, Jan. 2019

[8] L. Alves, E. F. Cruz, and A. M. R. Da Cruz, "Tracing Sustainability Indicators in the Textile and Clothing Value Chain using Blockchain Technology," 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), pp. 1–7, Jun. 2022

[9] S. Kravenkit and C. So-In, "Blockchain-Based traceability System for product recall," IEEE Access, vol. 10, pp. 95132–95150, Jan. 2022

[10] T. K. Agrawal, J. Angelis, W. A. Khilji, R. Kalaiarasan, and M. Wiktorsson, "Demonstration of a blockchain-based framework using smart contracts for supply chain collaboration," *International Journal of Production Research*, vol. 61, no. 5, pp. 1497–1516, Mar. 2022

[11] M. Ashraf and C. Heavey, "A Prototype of Supply Chain Traceability using Solana as blockchain and IoT," *Procedia Computer Science*, vol. 217, pp. 948–959, Jan. 2023.

POSTER


INGREDIENT TRACING USING BLOCKCHAIN FOR RAPID PRODUCT RECALL
SCAN HERE
AND TRACK YOUR FOOD
1. Run the Nutriguard App and login via your Web3 wallet
2. Scan the product QR Code
3. Check the product detail and register to your account to receive notification