**GreenTrack System Design**

**System Architecture for Green Track Blockchain-Based Supply Chain App**

System Architecture defines how different components of Green Track chain application interact. Using Ethereum blockchain, this architecture ensures data transparency, security, and immutability while allowing users to verify product origins.

The system consists of four main layers:

1. **Frontend Layer (React Native / Web App)**

* User-friendly mobile and web interface.
* QR code scanning for product verification.
* Displays product details from the blockchain.

1. **Backend Layer (Node.js + Express.js)**

* API to handle user requests.
* Communicates with the blockchain.
* Stores non-essential metadata in a database.

1. **Blockchain Layer (Ethereum / Smart Contracts)**

* Stores immutable product data (e.g., origin, quality tests, ownership history).
* Ensures decentralized verification.
* Uses Ethereum Smart Contracts for data integrity.

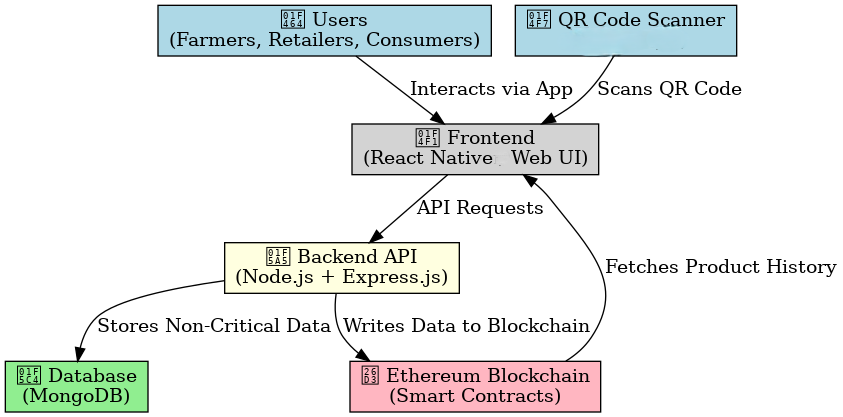
1. **Database Layer (MongoDB )**

* Stores user accounts, session data, and metadata.
* Stores references to blockchain transactions.

**System Components & Flow**

* Farmer brings produce to the collection point.
* Agency inspects the produce and enters data into the system.
* The backend API sends data to the Ethereum smart contract.
* A QR Code is generated and linked to the blockchain transaction.
* Retailers & Consumers can scan the QR code to verify the product’s origin and quality.
* Ownership transfer is recorded on the blockchain if the product is sold.

**System Architecture Diagram**



**Software Requirements**

**Blockchain Layer (Ethereum)**

Core Components:

- Ethereum Network (Public/Private)

- Public: Mainnet / Testnets (Ropsten, Rinkeby, Goerli)

- Private: Ganache (for local development)

- Smart Contracts (Solidity)

- Product registration

- Ownership transfer

- Quality verification

- Web3.js / Ethers.js (Blockchain interaction)

**Backend (Node.js & Express.js)**

Core Requirements:

- Node.js

- Express.js (REST API development)

- MongoDB (NoSQL database for off-chain data)

- Mongoose (ODM for MongoDB)

- JWT / OAuth 2.0 (Authentication)

API Endpoints Needed:

- User authentication (Farmers, Distributors, Retailers)

- Product registration & updates

- Blockchain transaction logging

- IoT sensor data integration

**Frontend (React.js)**

Core Requirements:

Key Frontend Modules:

- Dashboard (Track product journey)

- QR Code Scanner (Verify product authenticity)

- Admin Panel (Manage users, contracts)

- Mobile Responsiveness (For field agents)

**Hardware Requirements**

On-Farm & Warehouse Hardware

- Temperature & Humidity Sensors

- GPS Trackers (For logistics tracking)

- RFID / NFC Tags (Product identification)

Logistics & Transportation Hardware

- GPS Trackers (Real-time shipment tracking)

- Bluetooth Beacons (Warehouse proximity tracking)

- QR Code / NFC Labels (For product scanning)

3.3 Server & Node Infrastructure

Blockchain Nodes

- Minimum: 1-core CPU, 4GB RAM, 200GB SSD

- Recommended: Cloud-based (AWS EC2, Google Cloud)

Database & API Servers

- MongoDB Atlas (Managed cloud DB)

User Devices

- Farmers: Android/iOS smartphones (for data entry)

Functional & Non-Functional Requirements Document

Blockchain-Based Fresh Produce Supply Chain System (MVP)

# **Functional and Non-Functional Requirements**

## ****1. Functional Requirements****

### ****1.1. User Roles & Access****

* **Farmers** must be able to register produce batches, including details like origin, harvest date, and quantity.
* **Distributors** must be able to update shipment status (e.g., received, in transit, delivered).
* **Retailers** must be able to verify produce quality and update inventory records.
* **Consumers** must be able to scan a QR code to view the full history of a produce item.
* **Administrators** must have access to a dashboard for system monitoring and dispute resolution.

### ****1.2. Produce Registration & Tracking****

* The system must allow farmers to register fresh produce on the blockchain with a unique batch ID.
* Each registered produce item must generate a scannable QR code linked to its blockchain record.
* The system must store immutable records of:
  + Farm location and certification details.
  + Harvest and packaging dates.
  + Transportation logs (future phase with IoT integration).

### ****1.3. Pricing & Payments****

* The system must enforce transparent pricing using smart contracts to ensure farmers receive fair payments.
* Pricing must be based on predefined rules (e.g., market rates, organic certification premiums).
* Payments must be processed automatically via smart contracts upon delivery confirmation.
* Farmers must receive payments in cryptocurrency (ETH or stablecoins like USDT).

### ****1.4. Traceability & Consumer Access****

* Consumers must be able to scan a QR code to retrieve:
  + Farm origin and farmer details.
  + Harvest and expiration dates.
  + Supply chain journey (farm → distributor → retailer).
* The system must display this information in a user-friendly mobile and web interface.

### ****1.5. Dispute Resolution & Admin Controls****

* Administrators must be able to manually intervene in case of disputes (e.g., incorrect pricing, delivery issues).
* The system must log all changes made by administrators for audit purposes.

## ****2. Non-Functional Requirements****

### ****2.1. Performance & Scalability****

* The system must support at least **1,000 daily transactions** without significant delays.
* Blockchain data retrieval (e.g., QR code scans) must load within **2 seconds**.
* Smart contracts must be optimized for **low gas fees** to reduce operational costs.
* The system must be scalable to accommodate future IoT integrations (e.g., temperature sensors).

### ****2.2. Security & Data Integrity****

* All supply chain data must be stored immutably on the Ethereum blockchain.
* User wallets (MetaMask integration) must be securely authenticated.
* Sensitive off-chain data (e.g., farmer identity documents) must be encrypted and stored on IPFS.
* Smart contracts must undergo third-party audits to prevent vulnerabilities.

### ****2.3. Usability & Accessibility****

* The interface must be **mobile-responsive** for farmers and consumers in rural areas.
* The system must support **multiple languages** in future updates.
* The QR code scanning feature must work offline (cached data) in low-connectivity regions.

### ****2.4. Compliance & Regulations****

* The system must comply with **GDPR** for handling personal data (e.g., farmer identities).
* Smart contracts must align with **local agricultural trade laws**.
* All transactions must be auditable for regulatory reporting.

### ****2.5. Reliability & Fault Tolerance****

* The system must remain operational even if Ethereum network congestion occurs (fallback to Layer 2 solutions).
* Smart contracts must include fail-safes to prevent fund lockups due to errors.

## ****3. Key System Workflows****

### ****How Transparency is Achieved****

* Farmers log produce details → Stored permanently on Ethereum.
* Consumers scan QR codes → Retrieve blockchain records in real time.

### ****How Fair Pricing is Enforced****

* Smart contracts calculate prices based on predefined rules.
* Payments are locked in escrow and released automatically upon delivery confirmation.

### ****How Efficiency is Maintained****

* Gas-optimized smart contracts reduce transaction costs.
* Cached data ensures fast QR code scans even with slow internet.

## ****4. Future Enhancements (Post-MVP)****

* **IoT Sensors:** Monitor temperature/humidity during transport to reduce spoilage.
* **AI Grading:** Automatically classify produce quality using image recognition.
* **DeFi Loans:** Farmers access microloans backed by future produce sales.