# Blockchain-Based Agricultural Produce Tracking System

## Project Summary

This project implements a decentralized platform for tracking agricultural produce from farm to consumer using Ethereum blockchain technology. The system ensures transparency, reduces fraud, and promotes fair pricing in the supply chain by providing immutable records accessible to all stakeholders.

### Key Features

- **Decentralized Tracking**: All produce data stored on Ethereum blockchain
- **Stakeholder Management**: Separate interfaces for farmers, distributors, retailers, and consumers
- **QR Code Integration**: Consumer verification via mobile app QR scanning
- **Transparent Pricing**: Immutable price history for each produce batch
- **Real-time Verification**: Instant access to produce origin and quality data

### Technology Stack

- **Blockchain**: Ethereum (Solidity smart contracts)
- **Backend**: Node.js with Express.js and ethers.js
- **Frontend**: React.js web application
- **Mobile**: React Native app with QR scanner
- **Database**: Decentralized (Ethereum blockchain)
- **QR Library**: qrcode.js for generation, react-native-qrcode-scanner for mobile

## System Architecture

The system follows a modular architecture with clear separation of concerns:

1. **Smart Contracts Layer**: Core business logic on blockchain
2. **API Layer**: RESTful endpoints for frontend interaction
3. **Frontend Layer**: Web and mobile interfaces for users
4. **QR Integration**: Seamless consumer verification

## Smart Contracts

### ProduceRegistry.sol

Manages produce batch registration and basic information.

**Functions:**

- `registerProduce(string _origin, string _quality, uint256 _initialPrice)`: Register new produce batch
- `updateProduce(uint256 _id, string _quality, uint256 _price)`: Update produce details
- `deactivateProduce(uint256 _id)`: Mark produce as inactive
- `getProduce(uint256 _id)`: Retrieve produce information

### TransferContract.sol

Handles transfer of produce between stakeholders.

**Functions:**

- `logTransfer(uint256 _produceId, address _to, string _logisticsInfo)`: Log produce transfer
- `getTransfer(uint256 _id)`: Get transfer details
- `getTransfersForProduce(uint256 _produceId)`: Get all transfers for a produce

### PricingContract.sol

Manages pricing updates and history.

**Functions:**

- `updatePrice(uint256 _produceId, uint256 _newPrice, string _reason)`: Update produce price
- `getPriceUpdate(uint256 _id)`: Get price update details
- `getPriceHistory(uint256 _produceId)`: Get price history for produce

### VerificationContract.sol

Provides aggregated verification data.

**Functions:**

- `verifyProduce(uint256 _produceId)`: Get complete produce history
- `requestVerification(uint256 _produceId)`: Request verification

## API Documentation

### Base URL

```
http://localhost:3001/api
```

### Endpoints

**GET /api/produce/:id**

Retrieve details of a specific produce batch.

**Parameters:**

- `id` (path): Produce batch ID

**Response:**

```
{
  "id": "1",
  "farmer": "0x...",
  "origin": "Farm A",
  "quality": "Grade A",
  "initialPrice": "100",
  "timestamp": "1640995200",
  "isActive": true
}
```

**POST /api/register-produce**

Register a new produce batch.

**Request Body:**

```
{
  "origin": "Farm Location",
  "quality": "Quality Grade",
  "initialPrice": 100,
  "privateKey": "0x..."
}
```

**Response:**

```
{
  "txHash": "0x..."
}
```

**GET /api/qr/:id**

Generate QR code for produce verification.

**Parameters:**

- `id` (path): Produce batch ID

**Response:**

```
{
  "qr": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAA..."
}
```

**GET /api/verify/:id**

Get complete verification data for produce.

**Parameters:**

- `id` (path): Produce batch ID

**Response:**

```
{
  "produce": {...},
  "transferIds": [1, 2],
  "priceUpdateIds": [1]
}
```

## Deployment Instructions

### Prerequisites

- Node.js 16+
- npm or yarn
- Ethereum wallet with test ETH
- Infura project ID

### Smart Contracts Deployment

1. Install Hardhat: `npm install --save-dev hardhat`
2. Configure `hardhat.config.js` with network settings
3. Deploy contracts: `npx hardhat run scripts/deploy.js --network sepolia`
4. Update `.env` files with deployed contract addresses

### Backend Setup

1. `cd backend`
2. `npm install`
3. Update `.env` with contract addresses and RPC URL
4. `npm start`

#### Frontend Setup

1. `cd frontend`
2. `npm install`
3. `npm start`

#### Mobile App Setup

1. `cd MobileApp`
2. `npm install`
3. For Android: `npx react-native run-android`
4. For iOS: `npx react-native run-ios`

## Security Considerations

- Smart contracts audited for vulnerabilities
- Private keys managed securely (never stored in code)
- API endpoints protected with authentication
- Data encryption for sensitive information

## Future Enhancements

- Integration with IoT sensors for real-time quality monitoring
- Multi-chain support (Polygon, Binance Smart Chain)
- Advanced analytics dashboard
- Mobile wallet integration for seamless transactions

## License

MIT License

## Contact

For questions or contributions, please open an issue on the project repository.