Farm Inventory Management System (FIMS) - Complete Project Documentation

📋 PROJECT OVERVIEW

What is the Project?

The Farm Inventory Management System (FIMS) is a **premium, state-of-the-art full-stack web application** designed to bridge the gap between farmers and consumers in the Agri-Food supply chain. It provides a robust ecosystem for managing farm inventories, products, orders, and transactions with real-time tracking and secure payment integration.

Our Solution

FIMS solves the following problems:

- **Direct Farm-to-Consumer Connection**: Eliminates intermediaries
- **Inventory Management**: Real-time stock tracking with atomic operations
- **Farmer Verification**: Admin approval workflow for farmer registration
- **Secure Payments**: Multi-payment support (Razorpay online payments + Cash on Delivery)
- **Order Tracking**: Complete order lifecycle management
- **Stock Integrity**: Prevents over-ordering with atomic stock deduction

---

🛠️ TECHNOLOGY STACK

Frontend

- **Next.js 15.2.4** (App Router with TypeScript)
- **React 19.0.0** (Latest React version)
- **TailwindCSS 4** (Utility-first CSS)
- **DaisyUI 5.0.12** (Component library for Tailwind)
- **Tabler Icons** (@tabler/icons-react)
- **React Hot Toast** (Notifications)

Backend

- **Next.js API Routes** (Serverless API endpoints)

- **Node.js** (Runtime environment)
- **TypeScript** (Type safety)

Database

- **MongoDB** (NoSQL database)
- **Mongoose 8.13.2** (ODM for MongoDB)
- **Local MongoDB Compass** / Atlas Ready

Authentication & Security

- **JSON Web Tokens (JWT)** (jsonwebtoken ^9.0.2)
- **Bcrypt.js** (Password hashing)
- **Middleware-level Authorization** (Admin/Farmer/User roles)

Payment Integration

- **Razorpay 2.9.6** (Online payment gateway)
- **Cash on Delivery (COD)** support

Additional Technologies

- **Axios** (HTTP client)
- **Nodemailer** (Email notifications)
- **EJS** (Template engine for reports)
- **Ganache** (Blockchain testing - for future blockchain integration)
- **ESLint** (Code linting)
- **PostCSS** (CSS processing)

📁 FOLDER STRUCTURE

Code

```
-Farm-Inventory-Management-System/
│
├── public/                    # Static assets
│
├── src/                      # Source code
```

```
│   ├── app/                    # Next.js App Router
│   │   ├── (Home)/              # Home route group
│   │   ├── admin/               # Admin dashboard pages
│   │   ├── farmer/              # Farmer dashboard pages
│   │   ├── user/                # User/Consumer pages
│   │   ├── api/                 # API routes (Backend)
│   │   │   ├── admin/           # Admin API endpoints
│   │   │   ├── auth/            # Authentication endpoints
│   │   │   ├── helper/          # Helper API utilities
│   │   │   ├── inventory-logs/  # Inventory logging endpoints
│   │   │   ├── orders/          # Order management endpoints
│   │   │   ├── payment/         # Payment processing endpoints
│   │   │   ├── products/        # Product CRUD endpoints
│   │   │   └── user/            # User management endpoints
│   │   ├── globals.css          # Global styles
│   │   └── icon.png             # App favicon
│   │
│   ├── components/             # Reusable React components
│   │   ├── 404Image/            # 404 error component
│   │   ├── CameraFeed/          # Live camera integration
│   │   ├── Footer/              # Footer component
│   │   └── Navbar/              # Navigation component
│   │
│   ├── context/                # React Context for state management
│   │
│   ├── helper/                 # Helper utilities
│   │   └── reportTemplate.ejs   # Email/Report template
│   │
│   ├── middlewares/               # Middleware functions
│   │   └── db.config.ts           # MongoDB connection config
│   │
│   ├── models/                    # Mongoose data models
│   │   ├── User.ts                # User schema (Admin/Farmer/User)
│   │   □□□── Product.ts              # Product schema
│   │   ├── Order.ts               # Order schema
│   │   └── InventoryLog.ts        # Inventory tracking schema
│   │
│   └── types/                     # TypeScript type definitions
│
├── .gitignore                     # Git ignore rules
├── eslint.config.mjs              # ESLint configuration
├── image.png                      # Project screenshot
├── next.config.ts                 # Next.js configuration
├── package.json                   # Dependencies
├── package-lock.json              # Locked dependencies
├── postcss.config.mjs             # PostCSS configuration
├── README.md                      # Project documentation
├── tailwind.config.ts             # Tailwind CSS configuration
└── tsconfig.json                  # TypeScript configuration
```

🗄 DATABASE MODELS & SCHEMA

1. User Model (src/models/User.ts)

road2tec / -Farm-Inventory-Management-System / src / models / User.ts

```
import mongoose, { Schema } from "mongoose";


const UserSchema = new Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  contact: { type: String, required: true },
```

Purpose: Manages three types of users - Admin, Farmer, and Consumer (user)

2. Product Model (src/models/Product.ts)

road2tec / -Farm-Inventory-Management-System / src / models / Product.ts

```
const ProductSchema = new Schema({
  name: { type: String, required: true },
  description: { type: String, required: true },
  price: { type: Number, required: true },
  imageUrl: { type: String, required: true },
  category: { type: String, required: true },
```

Purpose: Stores farm products with inventory tracking capabilities

3. Order Model (src/models/Order.ts)

road2tec / -Farm-Inventory-Management-System / src / models / Order.ts

```
const OrderSchema = new Schema({
  userId: { type: Schema.Types.ObjectId, ref: "User", required: true },
  farmerId: { type: Schema.Types.ObjectId, ref: "User", required: true },
  products: [{
    product: { type: Schema.Types.ObjectId, ref: "Product" },
    quantity: { type: Number, required: true }
```

Purpose: Manages complete order lifecycle with payment and delivery tracking

4. InventoryLog Model (src/models/InventoryLog.ts)

Purpose: Tracks all inventory changes (stock additions/deductions) for audit trails

---

🏗️ SYSTEM ARCHITECTURE

Architecture Diagram

Code

```
┌──────────────────────────────────────────────────────────
──────────────────────────┐
│                CLIENT LAYER                  │
│  ┌──────────────┐   ┌──────────────┐   ┌──────────────┐
┌──────────────┐ │
│  │  Admin   │   │  Farmer  │   │  User   │   │  Camera  │ │
│  │Dashboard │   │Dashboard │   │ Shopping │   │   Feed   │ │
│  └───────┬──────┘   └───────┬──────┘   └───────┬──────┘
└─────┬───────┘ │
└───────────┼───────────────────┼───────────────────┼───────
───────────┼───────────┘
    │       │       │       │

└────────────────────┴────────────────────┴───────────────
┘
             │
    ┌──────────────────────────▼──────────────────────────┐
    │   NEXT.JS APP ROUTER (Frontend)    │
    │  - React 19 Components           │
    │  - TailwindCSS + DaisyUI          │
    │  - Client-side State (Context API)   │
    └─────────────────────┬─────────────────────────────┘
             │
    ┌─────────────────────▼─────────────────────────────┐
    │   NEXT.JS API ROUTES (Backend)     │
    │                     │
    │ /api/auth      - Authentication     │
    │ /api/admin      - Admin operations    │
    │ /api/products    - Product CRUD      │
    │ /api/orders     - Order management    │
    │ /api/payment    - Payment processing │
    │ /api/user       - User operations    │
    │ /api/inventory  - Inventory logs     │
    └───────────┬───────────────────┬───────────────────┘
            │           │
    ┌─────────▼─────────┐ ┌─────────▼─────────┐
    │ MIDDLEWARE        │ │ RAZORPAY API      │
    │                   │ │ (Payments)        │
    │ - db.config.ts    │ └───────────────────┘
    │ - Auth (JWT)      │
    │ - Role checks     │
    └─────────┬─────────┘
            │
    ┌─────────▼─────────────────────────────────────┐
    │      MONGODB DATABASE                         │
    │                                               │
    │ Collections:                                  │
    │ - users (Admin/Farmer/Consumer)               │
    │ - products (Farm inventory)                   │
    │ - orders (Transaction records)                │
    │ - inventorylogs (Audit trail)                 │
    └───────────────────────────────────────────────┘
```

🔄 WORKFLOW & DATA FLOW

1. User Registration & Authentication Flow

Code

```
[User Signs Up]
    ↓
```

[Choose Role: Admin/Farmer/Consumer]

↓

[Password Hashed (Bcrypt)]

↓

[Store in MongoDB - User Collection]

↓

[If Farmer → isApproved = false]

[If User/Admin → isApproved = true]

↓

[Admin Approves Farmer via Admin Dashboard]

↓

[JWT Token Generated on Login]

↓

[Token Stored in Context/Cookies]

↓

[Protected Routes Validated via Middleware]

2. Product Management Flow (Farmer)

Code

[Farmer Logs In]

↓

[Navigate to Farmer Dashboard]

↓

[Click "Add Product"]

↓

[Open Camera Feed Component]

↓

[Capture Product Image via Live Camera API]

↓

```
[Fill Product Details:
 - Name, Description, Price
 - Category, Stock, Unit
 - Harvest Date, Expiry Date
 - Organic Status]
     ↓
[POST /api/products/create]
     ↓
[Validate JWT + Farmer Role]
     ↓
[Save to MongoDB - Products Collection
 (with ownerId = Farmer's userId)]
     ↓
[Create InventoryLog Entry
 (action: "added", quantity: stock)]
     ↓
[Return Success + Display in Farmer Inventory]
```

3. Order Placement Flow (Consumer)

Code

```
[Consumer Browses Products]
     ↓
[Add Products to Cart]
     ↓
[Proceed to Checkout]
     ↓
[Enter Delivery Address]
     ↓
[Select Payment Method:
 - Online (Razorpay)
 - Cash on Delivery (COD)]
    ↓
┌───────────────────┴───────────────────┐
│              │
[IF ONLINE]     [IF COD]
    ↓             ↓
[POST /api/payment/create]  [POST /api/orders/create-cod]
    ↓             ↓
[Razorpay Order Created]   [Stock Check via findOneAndUpdate]
    ↓             ↓
[User Pays via Razorpay]   [Atomic Stock Deduction]
    ↓             ↓
[Webhook/Verify Payment]   [Order Created with:
    ↓            paymentStatus: "pending"
[POST /api/payment/verify]  paymentMethod: "cod"]
    ↓             ↓
[Atomic Stock Deduction]   [InventoryLog Created]
    ↓             │
[Order Created with:       │
 paymentStatus: "completed"  │
 paymentMethod: "online"]    │
    ↓             │
[InventoryLog Created]      │
     └───────────────┬───────────────┘
          ↓
[Order Saved in MongoDB - Orders Collection]
          ↓
```

```
[Notification Sent to Farmer]
              ↓
[User Sees Order in Order History]
```

4. Order Management Flow (Farmer)

Code

```
[Farmer Receives Order Notification]
     ↓
[View Order in Farmer Dashboard]
     ↓
[Update Delivery Status:
 - Pending → Shipped → Delivered]
     ↓
[PATCH /api/orders/update-status]
     ↓
[Update deliveryStatus in Order]
     ↓
[Consumer Sees Updated Status in Order History]
     ↓
[If COD: Farmer collects payment on delivery]
     ↓
[Farmer marks paymentStatus: "completed"]
```

5. Admin Management Flow

Code

```
[Admin Logs In]
     ↓
[Admin Dashboard Shows:
 - Pending Farmer Approvals
 - All Users
```

- All Products

- All Orders

- Real-time Analytics]

↓

[Admin Actions:

- Approve/Reject Farmers (PATCH /api/admin/approve-farmer)

- Delete Products (DELETE /api/admin/products/:id)

- Delete Users (DELETE /api/admin/users/:id)

- View System-wide Reports]

↓

[All Actions Logged in Database]

🔐 SECURITY FEATURES

1. **Password Security**: Bcrypt hashing with salt rounds
2. **JWT Authentication**: Stateless token-based auth
3. **Role-Based Access Control (RBAC)**: Admin/Farmer/User permissions
4. **Protected API Routes**: Middleware validates JWT and role
5. **Atomic Transactions**: MongoDB findOneAndUpdate prevents race conditions
6. **Input Validation**: Server-side validation for all inputs
7. **Secure Payment**: Razorpay signature verification

💳 PAYMENT INTEGRATION

Razorpay Integration

- **Create Order**: POST /api/payment/create generates Razorpay order
- **Verify Payment**: POST /api/payment/verify validates payment signature
- **Webhook Support**: Real-time payment status updates

COD Integration

- Instant order creation without payment gateway
- Stock deducted immediately to prevent overselling
- Payment marked as "pending" until delivery confirmation

📊 KEY FEATURES IMPLEMENTATION

1. Stock Guard System

TypeScript

```
// Atomic stock deduction to prevent over-ordering
const product = await Product.findOneAndUpdate(
  { _id: productId, stock: { $gte: quantity } },
  { $inc: { stock: -quantity } },
  { new: true }
);

if (!product) {
  throw new Error("Insufficient stock");
}
```

2. Camera Feed Integration

- Uses browser's navigator.mediaDevices.getUserMedia() API
- Live camera preview for product image capture
- Converts captured image to base64 for upload

3. Inventory Logging

- Every stock change creates an InventoryLog entry
- Tracks: action type (added/sold), quantity, timestamp, product reference
- Enables audit trails and analytics

🎨 UI/UX COMPONENTS

Components Overview

1. **Navbar**: Role-based navigation (Admin/Farmer/User views)
2. **Footer**: Site-wide footer with links
3. **CameraFeed**: Live camera integration for product images
4. **404Image**: Custom 404 error page
5. **Dashboard Cards**: DaisyUI cards for analytics
6. **Form Components**: Styled input fields with validation

Color Scheme

- **Admin Actions**: Emerald (approve) / Rose (reject)
- **Primary**: Tailwind's default primary colors
- **Status Indicators**:
    - Green (delivered/completed)
    - Yellow (pending/shipped)
    - Red (cancelled/failed)

📈 NO MACHINE LEARNING MODELS

Important Note: This project is **NOT** a machine learning or AI project. It's a traditional **full-stack MERN/MEAN application** focused on:

- CRUD operations
- Real-time inventory management
- Payment processing
- Order tracking
- User authentication

There are NO:

- Prediction algorithms
- ML models
- AI-based recommendations
- Data science components
- Training datasets

The **Ganache** dependency suggests potential **future blockchain integration** for supply chain traceability, but it's not currently implemented.

🚀 HOW THE SYSTEM WORKS (END-TO-END)

1. **Admin** sets up the system and approves farmers
2. **Farmers** register, get approved, and add products with live camera images
3. **Consumers** browse products, add to cart, and checkout
4. **Payment** processed via Razorpay (online) or marked for COD
5. **Stock** automatically deducted using atomic operations
6. **Orders** appear in both farmer dashboard (to fulfill) and user dashboard (to track)
7. **Farmers** update delivery status as they process orders
8. **InventoryLogs** maintain complete audit trail of all stock changes
9. **Admin** monitors entire system via analytics dashboard

🔗 FILE CONNECTIONS

Code

User Registration → User Model → MongoDB → JWT Token

↓

Product Addition → Product Model → Camera Component → MongoDB

↓

Order Placement → Order Model → Payment API → Stock Update → InventoryLog

↓

Admin Dashboard → All Models → Analytics & Reports

📦 DEPENDENCIES EXPLAINED

road2tec / -Farm-Inventory-Management-System / package.json

{

```
 "dependencies": {
   "@tabler/icons-react": "Icons for UI",
   "axios": "HTTP requests to APIs",
   "ejs": "Email/Report templates",
   "next": "Full-stack framework",
```

🎯 CONCLUSION

This is a **production-ready, full-stack e-commerce platform** specifically designed for the agricultural sector. It emphasizes:

- **Security** (JWT, bcrypt, atomic transactions)
- **Scalability** (Next.js, MongoDB, serverless API)
- **User Experience** (Modern UI with Tailwind + DaisyUI)
- **Real-time Operations** (Stock management, order tracking)
- **Payment Integration** (Razorpay + COD)

The system successfully connects farmers directly to consumers while maintaining data integrity, security, and a seamless user experience across all three user roles (Admin, Farmer, Consumer).