**Agribusiness Market Matching Platform**

## **Research Introduction**

## The Agribusiness Market Matching Platform is a web-based system developed using Laravel (PHP), MySQL, and HTML/CSS to connect farmers directly with buyers such as wholesalers, retailers, exporters, and processors.

## This system aims to reduce the dependency on middlemen by providing a transparent, efficient, and fair online marketplace for agricultural products. Farmers can list their harvests, buyers can post their product demands, and the platform’s matching engine automatically pairs both sides based on product type, quantity, and location.

## Through its dashboards and analytics tools, the system improves decision-making, enhances market access, and supports a more sustainable agricultural trade network for local communities.

### **Objectives**

1. To connect farmers directly with potential buyers through a centralized online marketplace.
2. To reduce middlemen costs and ensure fair, transparent agricultural pricing.
3. To automate product-demand matching using predefined conditions such as commodity, quantity, and location.
4. To provide dashboards and analytics for monitoring transactions, income, and demand trends.
5. To improve logistics coordination and tracking through integrated delivery management modules.

### 

### 

### **System Features**

The Agribusiness Market Matching Platform includes several core modules that enhance usability, efficiency, and transparency across all user roles.

#### 1. Dashboard and Analytics View

* Separate dashboards for Farmers, Buyers, Admins, and Logistics Providers.
* Summarized statistics and visual charts displaying total transactions, demand trends, and farmer income summaries.
* Admin dashboard includes report generation and system monitoring tools.

#### 2. Data Visualization Graphs and Charts

* Built using Chart.js or Laravel Charts.
* Displays graphical insights such as sales trends, product popularity, and regional demand.
* Helps users make data-driven decisions for production and purchasing.

#### 3. Tables and Tabular Data Presentations

* Transaction history tables showing order details, payment status, and delivery progress.
* User management tables for admins to verify accounts and monitor activity.
* Product and demand listing tables with search and filter features for easy access.

#### 4. User Registration and Login

* Multi-role authentication system for Farmers, Buyers, Admins, and Logistics Providers.
* Developed using Laravel Breeze or custom authentication with session-based access control.
* Role-based dashboard redirects after successful login.
* KYC (Know Your Customer) verification status integrated into profiles.

#### 5. Transaction and Record Management Modules

* Farmers can create, update, and delete product listings.
* Buyers can post demands and initiate transactions.
* A Matching Engine automatically connects related listings and demands.
* A Negotiation Module provides a built-in chat feature for communication between users.
* Transaction Table stores deal records, payment statuses (“Pending”, “Paid”), and delivery information (“Scheduled”, “Delivered”).
* Logistics Module records routes, delivery costs, and tracking details for transparency.

## 

## 

## 

## 

## 

## 

## 

## 

## **Full AI Prompt for System Prototype Development**

“Develop a complete working system prototype titled ‘Agribusiness Market Matching Platform’ using Laravel (PHP), MySQL, HTML/CSS, and optionally Bootstrap or TailwindCSS.

The purpose of this system is to connect farmers directly with buyers (wholesalers, retailers, processors, exporters) to reduce middlemen, ensure fair pricing, and improve transparency in agricultural transactions.

### 🔹 1. System Overview

The platform allows:

* Farmers to register, create profiles, and post products for sale.
* Buyers to register, post product demands, and find matching suppliers.
* The system to automatically match farmers’ products with buyers’ demands.
* Admins to manage users, monitor transactions, and generate reports.
* Logistics providers to track deliveries and update delivery status.

### 🔹 2. Required Features / Modules

A. Authentication & Role Management

* Multi-role registration & login (Farmer, Buyer, Admin, Logistics).
* Use Laravel Breeze or custom authentication.
* Role-based dashboard redirects.

B. Farmer Dashboard

* CRUD operations for product listings (product name, quantity, price, harvest date, etc.).
* View incoming buyer interests or matches.
* Update product availability.

C. Buyer Dashboard

* CRUD for posting product demands (product type, quantity, delivery location, target price).
* See matching products from farmers based on commodity and location.
* Start negotiations or transactions.

D. Matching Engine

* Automatically match farmer products and buyer demands using these conditions:  
  + Same product name (or related keyword)
  + Sufficient quantity
  + Same or nearby location
* Show possible matches in both buyer and farmer dashboards.

E. Negotiation and Transaction Module

* Simple chat or message log between buyer and farmer (store in database).
* When a deal is confirmed, record it in the transactions table.
* Include payment status (“Pending”, “Paid”) and delivery status (“Scheduled”, “Delivered”).

F. Logistics Module

* Assign logistics provider to a transaction.
* Record route, delivery cost, and tracking info.
* Update delivery status to “In Transit” or “Completed”.

G. Analytics Dashboard

* Use Chart.js or Laravel Charts to visualize:  
  + Total transactions, sales trends, and demand analytics.
  + Farmer income summary.
  + Popular commodities and locations.
* Include table views for all transaction records.

H. Admin Dashboard

* Manage and verify user accounts (Farmer, Buyer, Logistics).
* Monitor product listings, demands, and matches.
* Generate reports on total users, total sales, top products, and trends.

I. Additional Features

* Include KYC verification status in user profiles.
* Add support modules for microloans, insurance, and farming tips (basic CRUD info only).
* Use Blade templates for views and Bootstrap/Tailwind for styling.

### 🔹 3. Database Schema (MySQL)

Create the following tables with migrations or SQL scripts:

users(id, name, email, password, role, contact, location, kyc\_status, created\_at, updated\_at)

products(id, farmer\_id, product\_name, quantity, unit, price, harvest\_date, status)

demands(id, buyer\_id, product\_name, quantity, location, delivery\_date)

matches(id, product\_id, demand\_id, status, matched\_date)

transactions(id, match\_id, amount, payment\_status, delivery\_status)

logistics(id, provider\_id, match\_id, route, cost, tracking\_info)

messages(id, sender\_id, receiver\_id, message\_text, timestamp)

Include foreign key relationships for farmer\_id, buyer\_id, and match\_id accordingly.

### 🔹 4. Technology Stack

* Backend: Laravel (latest version)
* Database: MySQL
* Frontend: HTML, CSS, Blade templates, Bootstrap/TailwindCSS
* Analytics: Chart.js
* Payment Simulation: Simple status update (no real API needed)

### 🔹 5. Deliverables

Generate and show the following in the output:

1. Folder structure of the Laravel project (controllers, models, views, migrations).
2. Example Laravel routes (web.php) for each module.
3. Controller methods for user registration, product CRUD, demand CRUD, and matching logic.
4. Example Blade view files for:  
   * Login/Register
   * Farmer Dashboard
   * Buyer Dashboard
   * Admin Dashboard
   * Analytics Page
5. Example MySQL table creation scripts.
6. Seed sample data for 2 farmers, 2 buyers, and 2 transactions.
7. Instructions on how to run the project using artisan commands (php artisan migrate, php artisan serve).

### 🔹 6. Output Format

* Present code samples in organized blocks.
* Include explanations before each block.
* Use simple language for IT students.
* Avoid overly complex logic; focus on showing a working prototype structure.

### 🔹 7. Tone and Style

Write all explanations and comments in a beginner-friendly tone.  
 Keep the system realistic, technically correct, and consistent with Laravel conventions.”