# AGROLINK : BRIDGE BETWEEN CUSTOMERS AND FARMERS

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**


Submitted by

**Md. Afthab Raza Hussain, AP23110010398 | J. Akhil Joy, AP23110010037**

**N Hemanth Sai, AP23110011391 | P Surya, AP23110010788**



Under the Guidance of

**Mr. Yatharth sir**

**Assistant Professor, Department of CSE**


**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**


**[December, 2025]**

# Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success.

I am extremely grateful and express my profound gratitude and indebtedness to my project guide, **Mr. Yatharth**, Department of Computer Science & Engineering, SRM University, Andhra Pradesh, for her kind help and for giving me the necessary guidance and valuable suggestions in completing this project work.

Md. Afthab Raza Hussain, AP23110010398,

J. Akhil Joy, AP23110010037,

N Hemanth Sai, AP23110011391,

P Surya, AP23110010788

# Table of Contents


| | |
|---|---|
| Acknowledgements | 2 |
| Table of Contents | 3 |
| 1. Introduction | 4 |
| 2. Scenario-Based Intro | 5 |
| 3. Target Audience | 6 |
| 4. Project Goals and Objectives | 7 |
| 5. Key Features | 8 |
| 6. Pree-Requisites for Agrolink | 9 |
| 7. Project Structure | 11 |
| 8. Project Flow | 12 |
| 9. Project Execution | 20 |
| 10. Project Demo Links | 24 |

# 1. Introduction

Agriculture is one of the most important sectors of the economy, yet many farmers continue to face challenges in receiving fair prices for their produce due to the presence of multiple intermediaries. Customers, on the other hand, often struggle to access fresh, high-quality farm products at affordable rates. To address this gap, **Agrolink** is developed as a digital platform that directly connects farmers with customers, eliminating middlemen and ensuring transparency in the buying and selling process.

Agrolink empowers farmers by allowing them to **showcase, manage, and sell their produce online** at fair and profitable prices. Customers can browse a wide range of farm-fresh products, view detailed information, and purchase directly from verified farmers. This direct communication strengthens trust, enhances market accessibility, and promotes fairness in the agricultural trade ecosystem.

The project uses **React.js** for a dynamic and responsive user interface, **Tailwind CSS** for styling, and **JSON-Server** for simulating a backend database. The platform aims to support local farmers, reduce exploitation, and create a sustainable digital marketplace where both buyers and sellers benefit.

# 2. Scenario-Based Intro

Ravi, a small-scale farmer from a rural village, wakes up every morning before sunrise to tend to his crops. After months of hard work, he finally harvests his fresh tomatoes and takes them to the local market. But despite the quality of his produce, Ravi earns very little. Middlemen and wholesalers dictate the prices, leaving him with profits barely enough to support his family.

Meanwhile, in the nearby town, Priya wants to buy fresh and chemical-free vegetables for her family. She visits supermarkets and local vendors, but the prices are high, and the quality is often uncertain.
Both Ravi and Priya face problems—but their needs are perfectly aligned.

**What if Ravi could sell directly to Priya?**
No middlemen.
Fair prices.
Fresh produce.
Mutual trust.

This is where **Agrolink** comes into the picture.

Agrolink is a digital platform designed to **bridge the gap between farmers and customers**. It empowers farmers like Ravi to showcase their produce online, set fair prices, and reach a wider market. Customers like Priya can browse fresh farm products, view farmer profiles, and purchase directly with confidence.

Through this platform, Agrolink creates a **win-win ecosystem**: farmers earn better income, and customers receive high-quality, fresh produce at affordable rates. By eliminating intermediaries, Agrolink not only supports rural livelihoods but also brings transparency and efficiency to the agricultural supply chain.

# 3. Target Audience

Agrolink is designed to serve multiple groups of users who directly or indirectly participate in the agricultural value chain. The main target audiences include:

## 1. Farmers

This platform primarily targets small, medium, and large-scale farmers who want to:

- Sell their produce at fair prices
- Eliminate middlemen and market exploitation
- Reach customers directly
- Promote their farm products online
- Increase visibility and income

## 2. Customers / Consumers

Urban and rural customers who:

- Want to buy fresh, organic, and chemical-free produce
- Prefer direct purchase from farmers
- Look for affordable prices compared to supermarkets
- Want transparency about the source of products.

## 3. Local Retailers / Small Vendors

Retail shop owners who:

- Need reliable sources of fresh fruits, vegetables, and grains
- Want to buy directly in bulk from farmers
- Prefer stable quality and reasonable rates

# 4. Project Goals and Objectives

The primary goal of **Agrolink** is to create a seamless digital platform that empowers farmers and enables customers to access fresh, affordable produce directly from the source. Our objectives include:

**Fair Market Access:**

Provide farmers with a transparent platform where they can showcase and sell their products without middlemen, ensuring fair pricing and increased income.

### User-Friendly Experience:

Develop an intuitive and accessible interface that allows both farmers and customers to navigate, list, browse, and purchase products effortlessly, promoting trust and convenience.

### Efficient Product Management:

Enable farmers to easily add, update, and manage their produce listings with details such as price, quantity, category, and images—all in one place.

### Direct Buyer–Seller Interaction:

Facilitate smooth communication between customers and farmers through clear product details and farmer-specific information, ensuring transparency in the entire process.

### Modern Tech Stack:

Utilize advanced web development tools, including React.js, Tailwind CSS, and JSON Server, to deliver a fast, responsive, and scalable marketplace experience.

### Strengthening the Agricultural Ecosystem:

Support the growth of local agriculture by connecting rural farmers to urban markets, reducing dependency on intermediaries, and fostering sustainable development.

# 5. Key Features

**Farmer Dashboard**

Farmers can easily add, update, and delete product listings, manage prices, and track their available stock.

**Product Listing & Detailed View**

Customers can browse all products with images, descriptions, categories, and pricing details for better decision-making.

**Search & Filter Options**

Improves user experience by helping customers quickly find specific products based on category or name.

**Transparent Farmer Profiles**

Displays essential details such as farmer name, location, and contact information to build trust between buyers and sellers.

**Clean and Responsive UI**

Built using React.js and Tailwind CSS to provide a modern, responsive, and user-friendly interface across all devices.

**Lightweight Backend with JSON Server**

Uses JSON Server to simulate a backend system for storing and managing real-time product, farmer, and order data.

**Error-Free Navigation with React Router**

Ensures smooth transitions between pages like Home, Product Details, Dashboard, and Farmers page.

**Secure and Organized Data Handling**

All data dynamically loads from the backend, ensuring consistent product updates and reliable operation.

# 6. PRE-REQUISITES FOR AGROLINK

**Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows developers to execute JavaScript code outside the browser. It is essential for running development tools, package scripts, and local servers during frontend development.

For Agrolink, Node.js and npm are required to:

- Run the React development server
- Install project dependencies such as React Router and Tailwind CSS
- Operate JSON Server, which is used to simulate the backend for storing farmer and product data

Before building the Agrolink application, ensure that **Node.js and npm** are properly installed on your system.

- **React.js**:

  React.js is a widely used JavaScript library for building fast, interactive, and component-based user interfaces. For the Agrolink application, React.js enables the creation of reusable UI components such as product cards, farmer profiles, dashboards, and navigation elements. It ensures a smooth and responsive experience for both farmers and customers.

  To begin using React.js, follow these steps:

  - Create a new React app:

    ```
    npm create vite@latest
    ```

  Enter project name (e.g., **agrolink**) & select the preferred framework options.

  - Navigate to the project directory and install dependencies

    ```
    cd project-name

    npm install
    ```

- Running the React App:

  With the React app created, we can now start the development server and see React application in action.

- Start the development server:

  ```
  npm run dev
  ```

  This command starts the development environment, and you can view the Agrolink application in your browser at: http://localhost:5174/

- **HTML, CSS, and JavaScript**: Fundamental knowledge of HTML, CSS, and JavaScript is required to structure pages, create styles, handle component logic, and manage interactivity throughout the Agrolink application.

- **JSON Server:** Agrolink uses JSON Server to simulate backend. It stores data for:
- Farmer profiles
- Product listings
- Orders

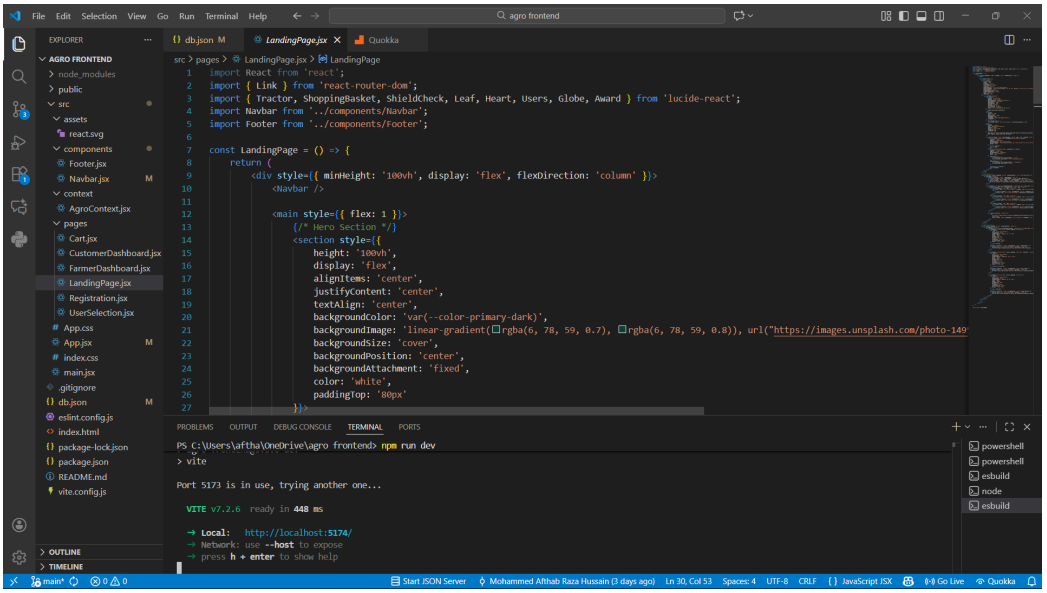
JSON Server allows frontend developers to work with real-time data without needing a fully built backend during development.

Run JSON Server using: `npx json-server --watch db.json --port 5000`

- **Version Control**: Git enables tracking changes, maintaining different versions, and collaborating with team members during Agrolink development. https://git-scm.com/downloads

- **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that supports efficient React development. Common choices include:
  •VisualStudio Code: Download from https://code.visualstudio.com/download
  • Sublime Text: Download from https://www.sublimetext.com/download
  •WebStorm:Download from https://www.jetbrains.com/webstorm/download

# 7. Project Structure




In the Agrolink project, the folders and files are structured to ensure modularity, clarity, and smooth integration between the frontend interface and the simulated backend.Important Files in our Agrolink project:

- **index.html** → The root HTML file where the React application is mounted.
- **package.json** → Contains all major dependencies such as React, React Router, Tailwind CSS, and development tools.
- **db.json** → Stores farmer information, product details, and orders using JSON Server to simulate a backend.

agrolink/

├── public/

├── src/

│ ├── components/ → Reusable UI elements (Navbar, Footer, ProductCard, etc.)

│ ├── pages/ → Screens such as Home, Farmers, Dashboard, Product Details

│ ├── api/ → Handles API calls to JSON Server (fetch products, farmers)

│ ├── App.jsx → Root component holding routes and overall layout

│ └── main.jsx → Entry point that renders the React application

├── db.json → Fake backend containing farmers, products, and orders

├── index.html → Base HTML file where  resides

├── package.json → Contains dependencies and project metadata

└── vite.config.js → Configuration file for Vite build tool

# 8. Project Flow

## Project demo:

Before starting to work on this project, let's see the demo.

https://github.com/mohammedafthabrazahussain/AGROLINK-FRONTEND

### Agrolink User Flow

1. User opens the Agrolink application**.**
2. The system fetches **products, farmers, and order data** from JSON Server.
3. The homepage displays all farm products using attractive product cards.
4. The user clicks a product to view **detailed information**, including price, quantity, description, and farmer details.
5. Farmers can log in to their dashboard to **add, edit, or delete** product listings.
6. Customers can browse farmer profiles and directly access produce details.
7. All data updates are reflected instantly from JSON Server.
8. The entire UI adapts responsively for mobile, tablet, and desktop screens using TailwindCSS.

### MILESTONE 1: Project Setup & Configuration

### 1. Install Required Tools

Open the project folder and install the necessary libraries:

- **React.js** – UI framework
- **React Router** – Page navigation
- **Axios / Fetch API** – API data handling
- **React Toastify** – Notifications
- **JSON Server** – Local backend for storing farmers/products
- **Tailwind CSS** – Responsive UI styling

All installed libraries will appear inside **package.json**

## 2. Recommended Documentation

- **React Installation →** https://react.dev/learn/installation
- **React Router →** https://reactrouter.com
- **TailwindCSS →** https://tailwindcss.com/docs/installation
- **JSON Server →** https://www.npmjs.com/package/json-server

# MILESTONE 2: Web Development

## 1. Setup React Application

- Create a new React app using Vite
- Install required dependencies (React Router, Toastify, Tailwind, JSON Server)
- Configure routing for all pages
- Set up JSON Server backend (db.json)
- Establish API connection using Axios or Fetch
- App.jsx:


## 2. Design UI Components

Develop reusable components for the Agrolink UI:

- Navbar
- Footer
- ProductCard
- FarmerCard
- Dashboard components
- Forms for Add/Edit Product

**Build pages:**

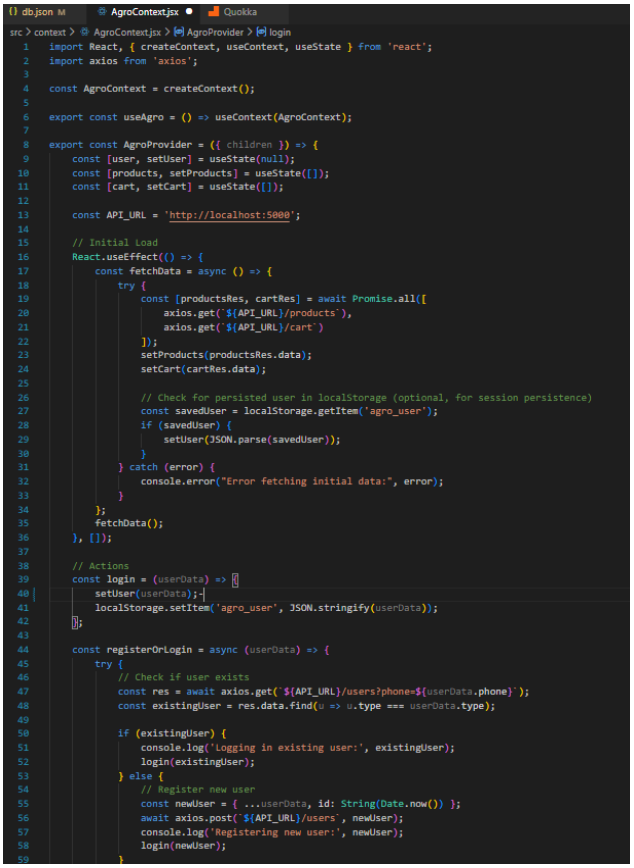
- Home Page
- Farmers Page
- Product Detail Page
- Farmer Dashboard
- Add Product Page
- Edit Product Page

Use TailwindCSS for layout, spacing, responsiveness, and styling.

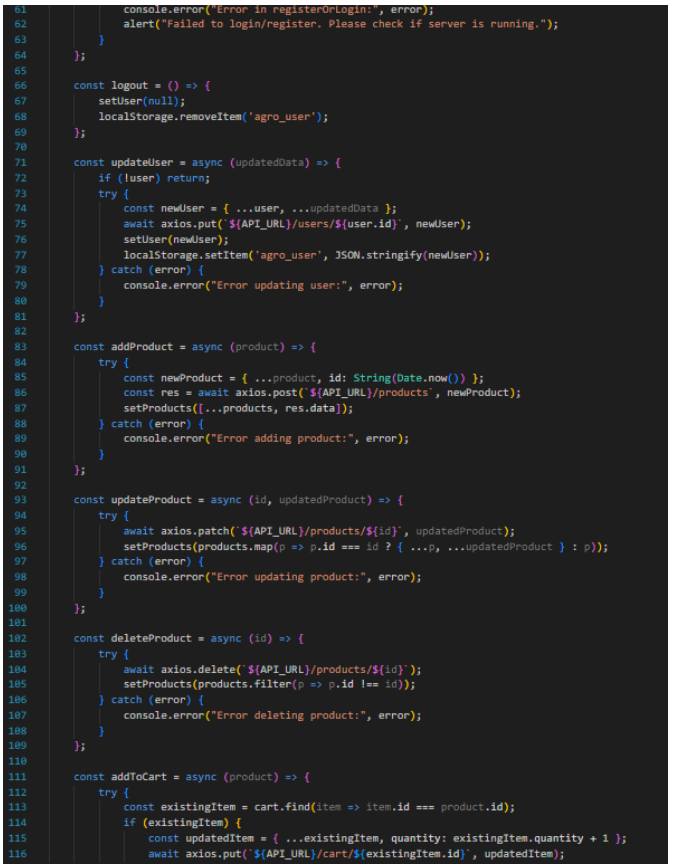
### 3. Implement Frontend Logic

- Fetch product and farmer data from JSON Server
- Display product list dynamically
- Handle CRUD operations:
    - Add product
    - Edit product
    - Delete product
- Bind backend data into UI components
- Implement toast notifications for actions like “Product Added”, “Updated”, “Deleted”
- Add search and filtering options (optional enhancement)

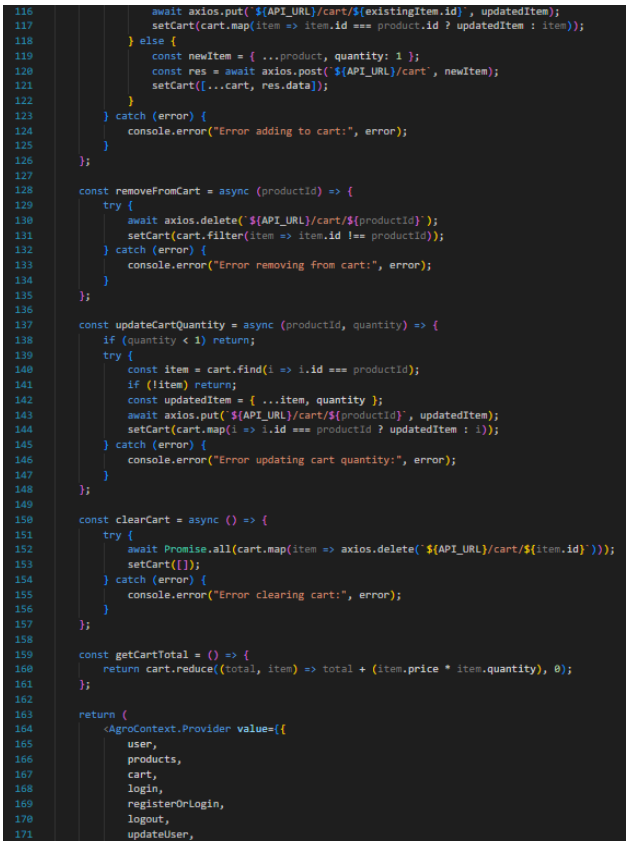
# Data Provider:



```
{} db.json M    AgroContext.jsx ●    Quokka
src > context > AgroContext.jsx > AgroProvider > login
import React, { createContext, useContext, useState } from 'react';
import axios from 'axios';

const AgroContext = createContext();

export const useAgro = () => useContext(AgroContext);

export const AgroProvider = ({ children }) => {
    const [user, setUser] = useState(null);
    const [products, setProducts] = useState([]);
    const [cart, setCart] = useState([]);

    const API_URL = 'http://localhost:5000';

    // Initial Load
    React.useEffect(() => {
        const fetchData = async () => {
            try {
                const [productsRes, cartRes] = await Promise.all([
                    axios.get(`${API_URL}/products`),
                    axios.get(`${API_URL}/cart`)
                ]);
                setProducts(productsRes.data);
                setCart(cartRes.data);

                // Check for persisted user in localStorage (optional, for session persistence)
                const savedUser = localStorage.getItem('agro_user');
                if (savedUser) {
                    setUser(JSON.parse(savedUser));
                }
            } catch (error) {
                console.error("Error fetching initial data:", error);
            }
        };
        fetchData();
    }, []);

    // Actions
    const login = (userData) => {
        setUser(userData);
        localStorage.setItem('agro_user', JSON.stringify(userData));
    };

    const registerOrLogin = async (userData) => {
        try {
            // Check if user exists
            const res = await axios.get(`${API_URL}/users?phone=${userData.phone}`);
            const existingUser = res.data.find(u => u.type === userData.type);

            if (existingUser) {
                console.log('Logging in existing user:', existingUser);
                login(existingUser);
            } else {
                // Register new user
                const newUser = { ...userData, id: String(Date.now()) };
                await axios.post(`${API_URL}/users`, newUser);
                console.log('Registering new user:', newUser);
                login(newUser);
            }
```



```
            console.error("Error in registerOrLogin:", error);
            alert("Failed to login/register. Please check if server is running.");
        }
    };

    const logout = () => {
        setUser(null);
        localStorage.removeItem('agro_user');
    };

    const updateUser = async (updatedData) => {
        if (!user) return;
        try {
            const newUser = { ...user, ...updatedData };
            await axios.put(`${API_URL}/users/${user.id}`, newUser);
            setUser(newUser);
            localStorage.setItem('agro_user', JSON.stringify(newUser));
        } catch (error) {
            console.error("Error updating user:", error);
        }
    };

    const addProduct = async (product) => {
        try {
            const newProduct = { ...product, id: String(Date.now()) };
            const res = await axios.post(`${API_URL}/products`, newProduct);
            setProducts([...products, res.data]);
        } catch (error) {
            console.error("Error adding product:", error);
        }
    };

    const updateProduct = async (id, updatedProduct) => {
        try {
            await axios.patch(`${API_URL}/products/${id}`, updatedProduct);
            setProducts(products.map(p => p.id === id ? { ...p, ...updatedProduct } : p));
        } catch (error) {
            console.error("Error updating product:", error);
        }
    };

    const deleteProduct = async (id) => {
        try {
            await axios.delete(`${API_URL}/products/${id}`);
            setProducts(products.filter(p => p.id !== id));
        } catch (error) {
            console.error("Error deleting product:", error);
        }
    };

    const addToCart = async (product) => {
        try {
            const existingItem = cart.find(item => item.id === product.id);
            if (existingItem) {
                const updatedItem = { ...existingItem, quantity: existingItem.quantity + 1 };
                await axios.put(`${API_URL}/cart/${existingItem.id}`, updatedItem);
```



```
                setCart(cart.map(item => item.id === product.id ? updatedItem : item));
            } else {
                const newItem = { ...product, quantity: 1 };
                const res = await axios.post(`${API_URL}/cart`, newItem);
                setCart([...cart, res.data]);
            }
        } catch (error) {
            console.error("Error adding to cart:", error);
        }
    };

    const removeFromCart = async (productId) => {
        try {
            await axios.delete(`${API_URL}/cart/${productId}`);
            setCart(cart.filter(item => item.id !== productId));
        } catch (error) {
            console.error("Error removing from cart:", error);
        }
    };

    const updateCartQuantity = async (productId, quantity) => {
        if (quantity < 1) return;
        try {
            const item = cart.find(i => i.id === productId);
            if (!item) return;
            const updatedItem = { ...item, quantity };
            await axios.put(`${API_URL}/cart/${productId}`, updatedItem);
            setCart(cart.map(i => i.id === productId ? updatedItem : i));
        } catch (error) {
            console.error("Error updating cart quantity:", error);
        }
    };

    const clearCart = async () => {
        try {
            await Promise.all(cart.map(item => axios.delete(`${API_URL}/cart/${item.id}`)));
            setCart([]);
        } catch (error) {
            console.error("Error clearing cart:", error);
        }
    };

    const getCartTotal = () => {
        return cart.reduce((total, item) => total + (item.price * item.quantity), 0);
    };

    return (
        <AgroContext.Provider value={{
            user,
            products,
            cart,
            login,
            registerOrLogin,
            logout,
            updateUser,
```

Creates an **AgroContext** using c**reateContext()** to hold global data such as user details, product listings, and cart information.

Defines an AgroProvider component that wraps the entire application, allowing all nested components to access shared data through the context.

Uses useState hooks to create and manage important global state variables:

- user → stores logged-in farmer/customer
- products → stores all product listings
- cart → stores items added by the user

Loads initial data using useEffect(), where it fetches products and cart items from JSON Server, and retrieves saved user information from localStorage.

Implements authentication functions, such as:

- login() → logs in existing user
- registerOrLogin() → registers a new user or logs in if already registered
- logout() → clears user session

Provides CRUD operations for products, including:

- addProduct()
- updateProduct()
- deleteProduct()

Manages customer cart functionality, offering:

- addToCart()
- removeFromCart()
- updateCartQuantity()
- clearCart()
- getCartTotal() → calculates total cost

Returns an AgroContext.Provider wrapping all child components, passing down state values and functions via the value prop.

Exports the useAgro() hook to easily access context data inside any component using useContext.

Wraps important components inside the AgroProvider, ensuring every page (Home, Dashboard, Product Details, Cart, Farmers) receives shared data without prop-drilling

## Code Component:-UI Components

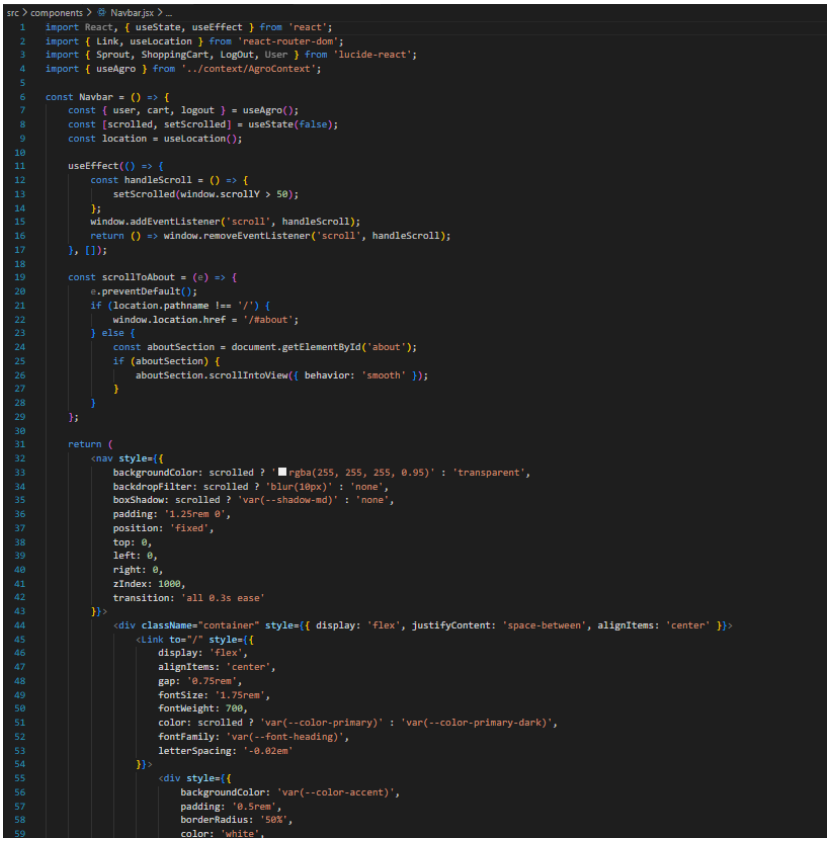
### 1. footer Component:


**Code Description:**

- **Imports React** to create the functional component.
- **Imports Link from React Router** to enable navigation between pages without reloading the browser.
- **Imports TailwindCSS classes** to style the navbar with spacing, colors, and layout.
- **Defines the Navbar functional component**, which serves as the top navigation bar for the Agrolink application.
  **Creates a navigation container** using a `<nav>` element styled with background color, padding, and flex layout for proper alignment.
- **Displays the Agrolink brand/title**, usually positioned on the left side of the navbar.
- **Renders navigation links** such as:
  - Home, Farmers
  - Dashboard, Cart
  - Login / Profile

  Each link is wrapped inside a Link component so users can navigate smoothly.
- **Applies hover effects and text styles** using Tailwind classes to improve the user experience and make navigation interactive.
- **Ensures the Navbar is responsive**, adjusting layout automatically for different screen sizes using utility classes.
- **Exports the Navbar component as the default export**, allowing it to be used inside `App.jsx` and displayed across all pages.

## 2. Navbar Component:



```jsx
import React, { useState, useEffect } from 'react';
import { Link, useLocation } from 'react-router-dom';
import { Sprout, ShoppingCart, LogOut, User } from 'lucide-react';
import { useAgro } from '../context/AgroContext';

const Navbar = () => {
    const { user, cart, logout } = useAgro();
    const [scrolled, setScrolled] = useState(false);
    const location = useLocation();

    useEffect(() => {
        const handleScroll = () => {
            setScrolled(window.scrollY > 50);
        };
        window.addEventListener('scroll', handleScroll);
        return () => window.removeEventListener('scroll', handleScroll);
    }, []);

    const scrollToAbout = (e) => {
        e.preventDefault();
        if (location.pathname !== '/') {
            window.location.href = '/#about';
        } else {
            const aboutSection = document.getElementById('about');
            if (aboutSection) {
                aboutSection.scrollIntoView({ behavior: 'smooth' });
            }
        }
    };

    return (
        <nav style={{
            backgroundColor: scrolled ? 'rgba(255, 255, 255, 0.95)' : 'transparent',
            backdropFilter: scrolled ? 'blur(10px)' : 'none',
            boxShadow: scrolled ? 'var(--shadow-md)' : 'none',
            padding: '1.25rem 0',
            position: 'fixed',
            top: 0,
            left: 0,
            right: 0,
            zIndex: 1000,
            transition: 'all 0.3s ease'
        }}>
            <div className="container" style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
                <Link to="/" style={{
                    display: 'flex',
                    alignItems: 'center',
                    gap: '0.75rem',
                    fontSize: '1.75rem',
                    fontWeight: 700,
                    color: scrolled ? 'var(--color-primary)' : 'var(--color-primary-dark)',
                    fontFamily: 'var(--font-heading)',
                    letterSpacing: '-0.02em'
                }}>
                    <div style={{
                        backgroundColor: 'var(--color-accent)',
                        padding: '0.5rem',
                        borderRadius: '50%',
                        color: 'white',
```

```jsx
                        display: 'flex'
                    }}>
                        <Sprout size={24} />
                    </div>
                    <span>AgroLink</span>
                </Link>

                <div style={{ display: 'flex', gap: '2.5rem', alignItems: 'center' }}>
                    <a href="#about" onClick={scrollToAbout} style={{ fontWeight: 600, color: 'var(--color-text-main)', fontSize: '0.95rem', textTransform: 'uppercase', letterSpacing: '0.05em', cursor: 'pointer' }}>About</a>

                    {user && (
                        <div style={{ display: 'flex', alignItems: 'center', gap: '1.5rem' }}>
                            <span style={{ fontWeight: 600, color: 'var(--color-primary)' }}>{user.name}</span>
                            {user.type === 'customer' && (
                                <Link to="/cart" style={{ position: 'relative', color: 'var(--color-text-main)' }}>
                                    <ShoppingCart size={24} />
                                    {cart.length > 0 && (
                                        <span style={{
                                            position: 'absolute',
                                            top: '-8px',
                                            right: '-8px',
                                            backgroundColor: 'var(--color-accent)',
                                            color: 'white',
                                            borderRadius: '50%',
                                            width: '20px',
                                            height: '20px',
                                            display: 'flex',
                                            alignItems: 'center',
                                            justifyContent: 'center',
                                            fontSize: '0.7rem',
                                            fontWeight: 'bold',
                                            boxShadow: '0 2px 4px rgba(0,0,0,0.2)'
                                        }}>
                                            {cart.reduce((acc, item) => acc + item.quantity, 0)}
                                        </span>
                                    )}
                                </Link>
                            )}
                            <button onClick={logout} style={{ background: 'none', color: 'var(--color-text-muted)', display: 'flex', alignItems: 'center', gap: '0.5rem' }}>
                                <LogOut size={20} />
                            </button>
                        </div>
                    )}
                </div>
            </div>
        </nav>
    );
};

export default Navbar;
```

## Code Description:

Imports React hooks (useState, useEffect) for managing scroll state and controlling behavior when the navbar becomes sticky.

Imports Routing utilities (Link, useLocation) from React Router, allowing navigation without page reloads and detecting the current route.

Imports icons (Sprout, ShoppingCart, LogOut, User) from lucide-react to visually enhance the navbar.

Imports the global AgroContext using useAgro() to access shared data such as:

- user → logged-in user details
- cart → items in user's cart
- logout → function to log out users

Creates a scrolled state using useState(false) to change navbar style (blur, shadow, background) when the user scrolls past 50px.

Adds a scroll listener inside useEffect to dynamically update the navbar's appearance based on scroll behavior.

Defines scrollToAbout function, which:

- Smoothly scrolls to the "About" section
- Redirects to home page first if the user is on another route

Returns a modern, fixed Navbar layout using inline CSS and Tailwind variables:

- Background dynamically changes based on scroll
- Navbar stays fixed at the top with smooth transitions
- Uses flexbox for alignment

Displays brand logo "AgroLink" using the Sprout icon, styled with custom color, spacing, and font.

Renders navigation links, including:

- "About" section link with smooth scroll animation.

Conditionally displays logged-in user controls, only if user exists:

- User name
- Cart icon for customers with live badge count (total quantity)
- Logout button with LogOut icon

Computes cart quantity using:

```
cart.reduce((acc, item) => acc + item.quantity, 0)
```
to show total number of items.

Applies clean UI styling, animations, spacing, hover effects, and shadow using Tailwind variables and inline CSS.

Exports the Navbar component for use across all pages via <Navbar />.

# 9. Project Execution

After successfully completing the development of the Agrolink platform, you can run the React project using the command:

npm run dev

This will start the Vite development server and launch the Agrolink frontend in your browser at: http://localhost:5174

Agrolink uses JSON Server as a lightweight backend for storing:

Farmer details
Customer details
Products
Cart items

To start the JSON Server, run:

npx json-server --watch db.json --port 5000

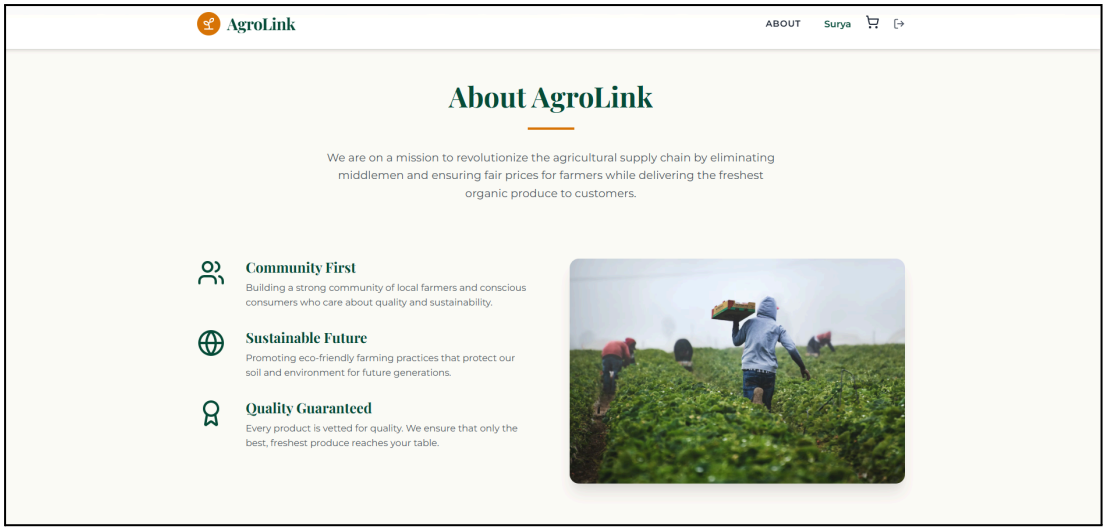
This will start the backend API at: http://localhost:5000

The API exposes useful endpoints such as:

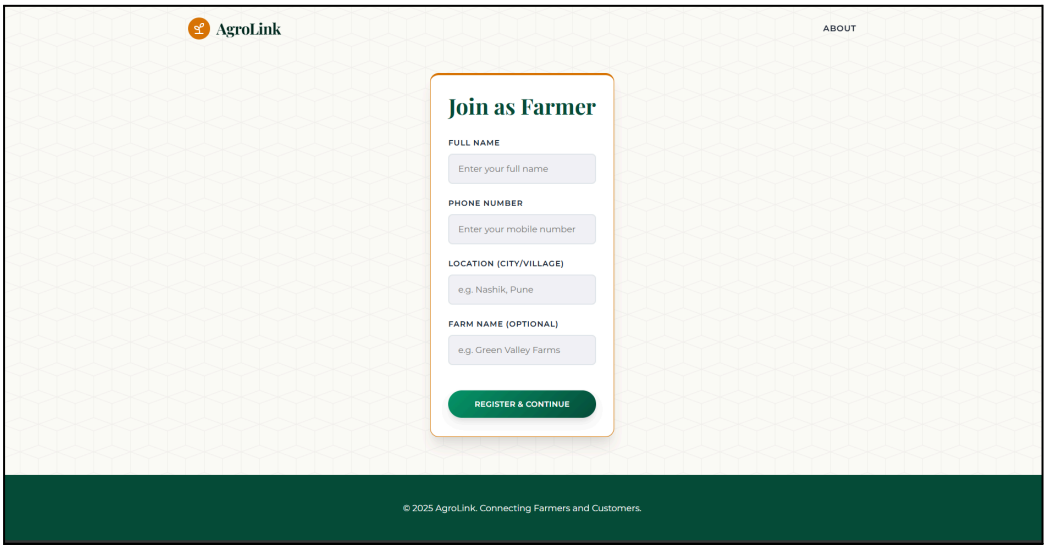
- /users
- /products
- /cart

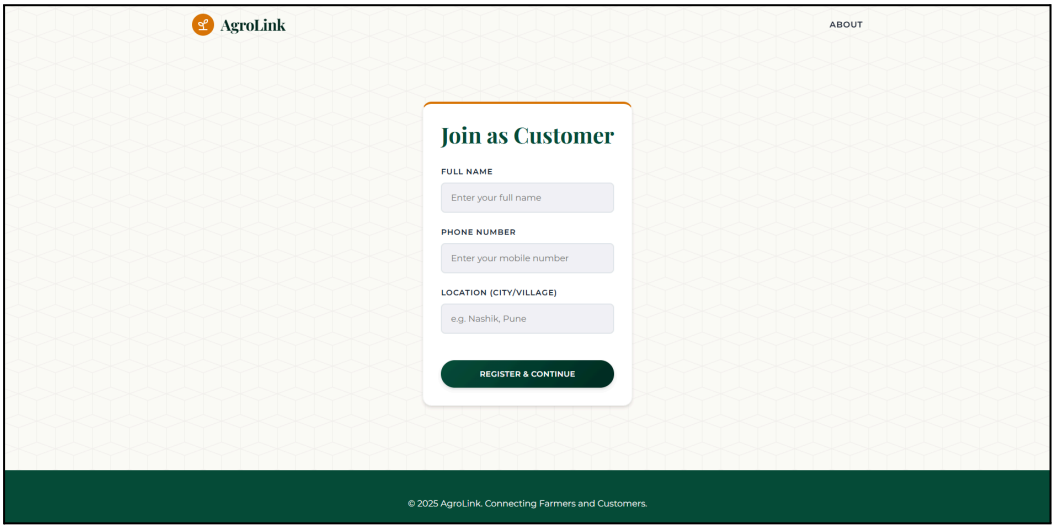
**Below are some screenshots representing the different components and pages of the application:**

**LandingPage Component:** The Landing Page welcomes users to Agrolink and provides quick access to customer and farmer sections with a clean, engaging interface

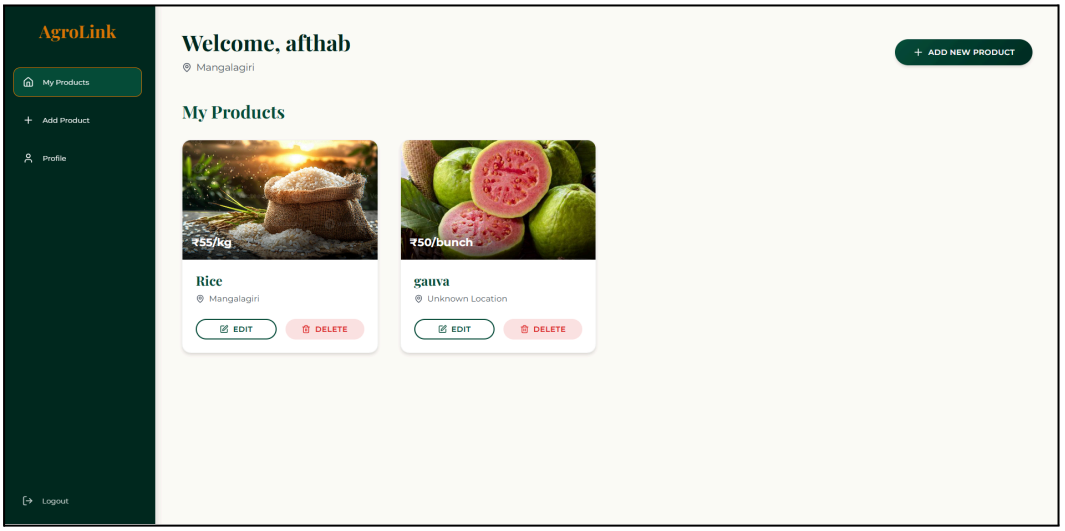


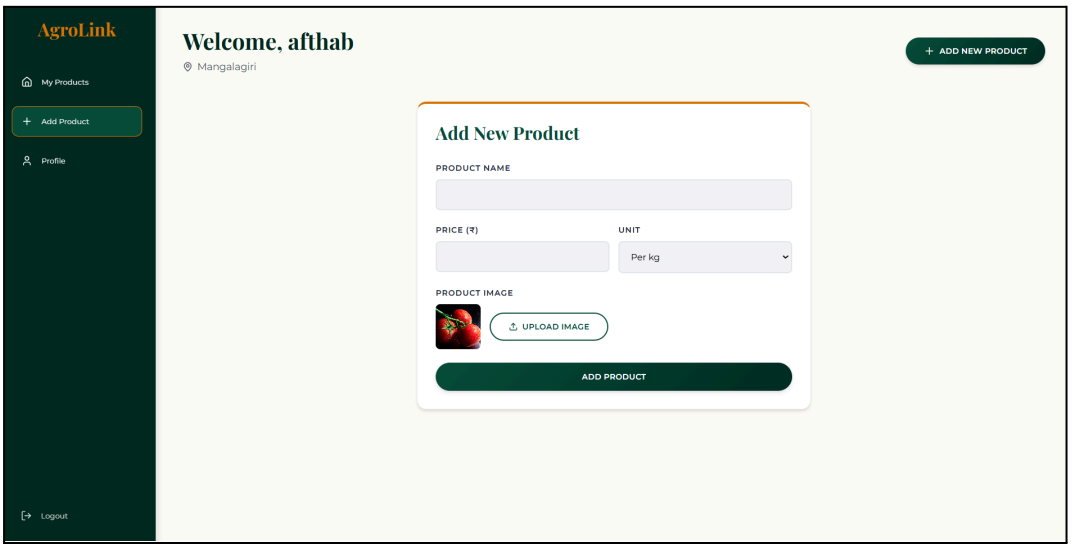

**Registration Component(Farmer,Customer)**: The Registration Page allows new farmers and customers to create an account by entering their basic details, which are stored through JSON Server.

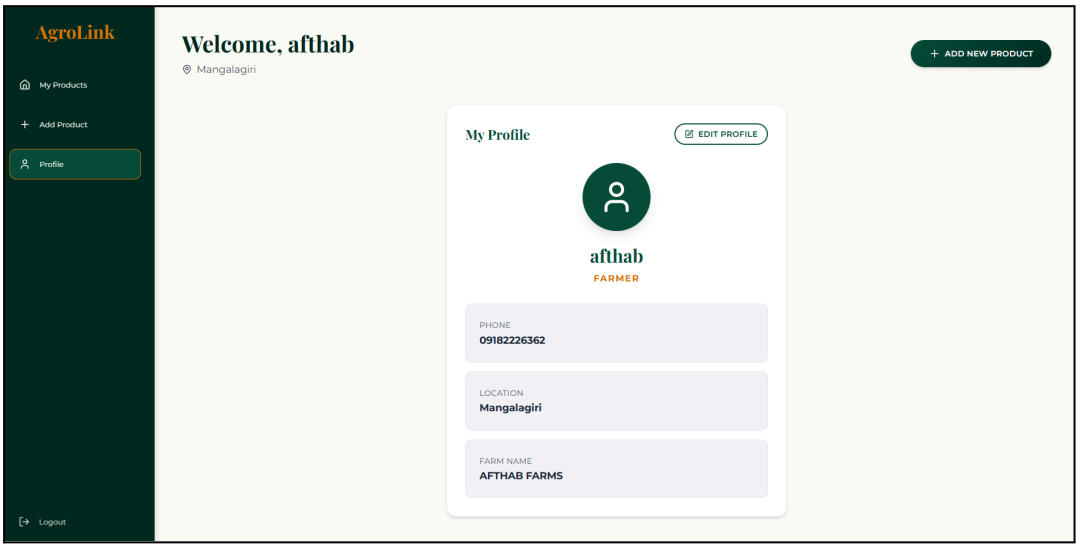


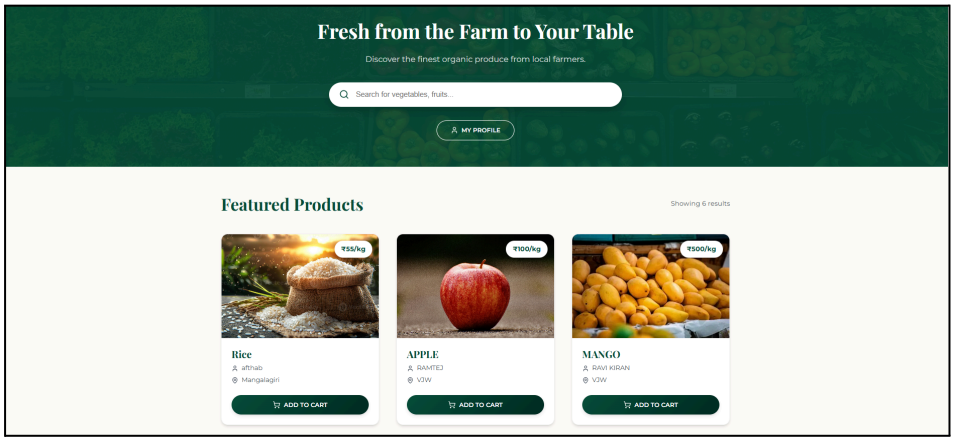
**Farmer Dashboard Component:** The Farmer Dashboard enables farmers to manage their products—add, edit, delete—and view all their listings in one place.

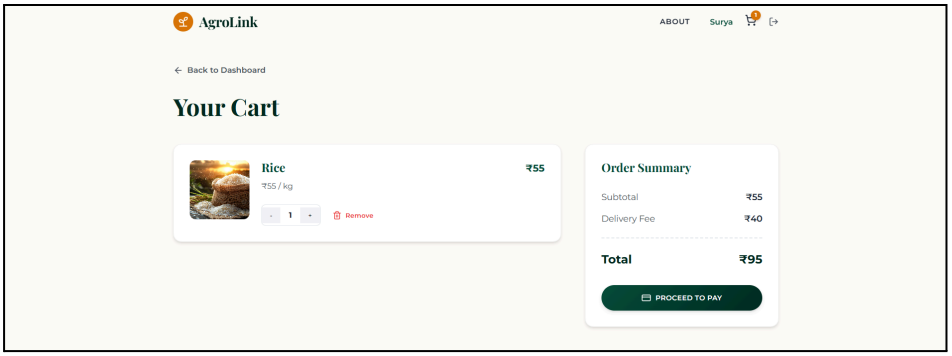


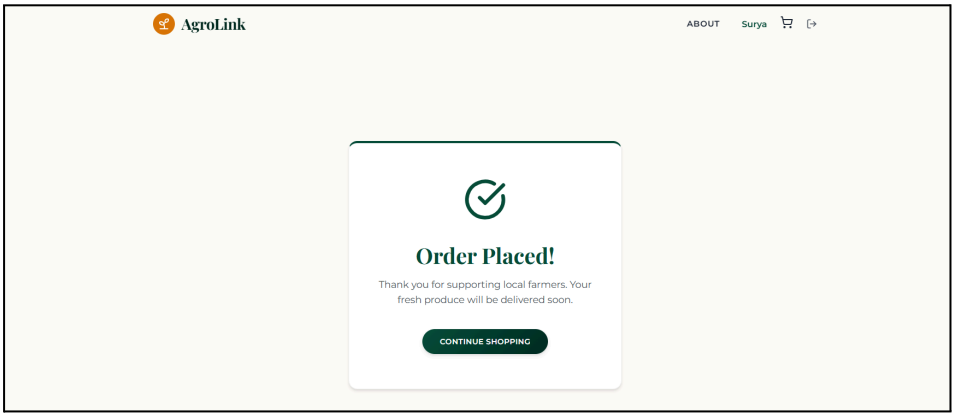



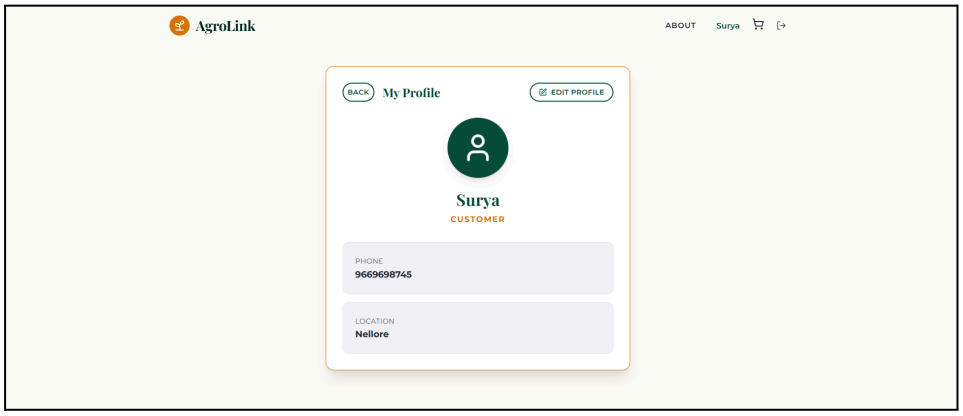



**Customer Dashboard Component:** The Customer Dashboard displays all available farm products, allowing customers to browse, search, and buy directly from farmers.



# 10. Project Demo Links

**Project Demo Explanation:**
https://drive.google.com/file/d/1Iafq9OM0LkmV6XTufuTdZhgHkwVW7T4u/view?usp=drive_link

**Project Code Explanation:**
https://drive.google.com/file/d/15q-5htwWOFzutZwB-7yJ0Hs2xxPKvEz_/view?usp=sharing

**Github Link:** https://github.com/mohammedafthabrazahussain/AGROLINK-FRONTEND