AgroConnect: A Mobile Application for Direct Market Access for Farmers Using Cloud-Based Architecture and Real-Time Demand Management

Dr. Sudhir Hate[1], Vedant Mallamvar[2], Amruta More[3], Snehal Fulari[4]
[1,2,3,4]*G H Raisoni College of Engineering & Management, Wagholi, Pune*

***Abstract*—The traditional agricultural supply chain consists of several tiers of middlemen and Indian farmers end up earning only 20-30% of final consumer price, leading to post-harvest losses of more than 1 crore annually. In this paper, we present AgroConnect, a cross-platform mobile application built with React Native and a scalable cloud-based backend (Express.js on the Bun runtime and PostgreSQL hosted on Neon DB, managed with Prisma ORM) that enables direct links between farmers and consumers, retailers and businesses, thereby removing middlemen. This system will support two type of user Farmer and Buyer. Farmers can post their produce live, upload images from Cloudinary, quote prices based on demand and integrate with UPI/COD/card payments. Buyers may search for products by location and follow the entire order cycle. Better Auth implements role-based access control (RBAC) using session tokens. The architecture is a client-server design with RESTful API communication and Zod-based input validation. For functional testing, there were 48 test cases, 97.9% pass rate, and verified correct on all modules like registration, produce listing, demand quoting, ordering and checkout. Fair, transparent and digitally enabled agricultural trade: An initiative called AgroConnect is a bi-partisan initiative designed to raise farmer income by 40-60% and reduce post-harvest losses by 30 per cent.**

***Index Terms*—Smart Agriculture, Farmer Marketplace, React Native, Agricultural E-commerce, Demand Forecasting, Precision Agriculture**

I. INTRODUCTION

Walk into any vegetable market in Maharashtra early in the morning and you will likely find a farmer who drove three hours before dawn, paid a toll, unloaded his produce at a mandi, waited for an arthiya to assign him a buyer, and walked home with barely forty rupees for every hundred the consumer paid. That gap—sixty rupees out of every hundred vanishing between farm gate and shopping basket—is not incidental. It is baked into a supply chain that was designed around physical proximity and information asymmetry, both of which are no longer necessary in an era of smartphones and cloud computing. India's agricultural census data [1] tells us that over 58% of the rural population still depends on farming, yet the sector's share of national income keeps shrinking relative to the effort invested. More than 86% of those farmers own less than two hectares [2], which means they have almost no bargaining power when they walk into a mandi where a handful of commission agents control the price.

The numbers behind this imbalance are striking. Farmers in India typically take home somewhere between 30% and 40% of what the end consumer eventually pays [2]. The rest gets carved up by traders, transporters, and commission agents who charge 6 to 10% on each transaction. Meanwhile, roughly a crore's worth of food rots every year simply because there was no quick way to find a buyer before the tomatoes turned soft or the mangoes bruised [3]. The government's response—the electronic National Agriculture Market or eNAM—was a genuine effort, but it assumed farmers had desktop computers, formal mandi registration, and enough English literacy to navigate a web portal. Most rural farmers have none of these things, and adoption has consequently stalled [4].

What farmers do have, increasingly, is a basic Android phone. Several studies have shown that even modest mobile connectivity shifts the balance of market information toward producers and away from traders [5][6]. Yet the mobile agricultural apps that exist today are largely list-and-browse tools: a farmer can post a

listing and wait, or a buyer can scroll through produce. What is missing is a two-sided negotiation layer—a mechanism where buyers can say "I need 300 quintals of wheat at this price by Friday" and farmers can respond competitively, much the way a reverse auction works. Existing platforms also tend to lump farmers and buyers into the same interface without role-based permissions, which creates both security gaps and usability confusion [7][8].

This paper describes how we built AgroConnect to close those gaps. The application is a cross-platform mobile tool developed using React Native, with a backend built on Express.js running on Bun and a PostgreSQL database hosted on Neon DB and managed through Prisma ORM. What distinguishes it from prior work is not any single flashy feature but rather the combination of things it gets right simultaneously: clean role separation between farmers and buyers enforced at the API level; a live demand-quoting workflow where buyers post requirements and farmers bid competitively; UPI, cash-on-delivery, and card payment support; geolocation-based product discovery; and an order tracking system that keeps both parties informed without requiring them to exchange phone numbers.

II. LITERATURE SURVEY

A. Foundational Research on ICT and Farm Income

The question of whether giving farmers access to information technology actually improves their income has been studied seriously since the early 2000s. Mittal and Mehar [11] gathered data from over 1,000 farm households across India and found that farmers who owned mobile phones received prices that were 8-12% closer to the prevailing wholesale rate than those who did not. The explanation is straightforward: a farmer who can call ahead to check the mandi price before travelling 50 kilometres is far less likely to accept a lowball offer from the local trader. Aker [16] reached similar conclusions in a study of extension services in Sub-Saharan Africa, where mobile-mediated agricultural advice improved knowledge uptake by about 22% compared with the traditional model of government field officers visiting villages monthly. Together, these two papers established that mobile connectivity is not a luxury for small farmers—it is an income-protection tool.

Birthal and colleagues [12] contributed a different but equally important angle. Their analysis of crop diversification patterns in India revealed that small farmers seeking to shift from staple grains to higher-value vegetables and fruits were often blocked not by agronomic knowledge gaps but by the lack of reliable market channels. Without an assured buyer who would pay a fair price for perishable produce, the rational choice was to stick with wheat or rice, even if the soil and climate were better suited to something more profitable. Reardon and Timmer [13] placed this observation in a global context, noting that organised retail and institutional buyers in developing economies were increasingly willing to source directly from producer groups, but only when those groups could guarantee consistent quality and reliable delivery schedules. Both studies point toward the same conclusion: what rural farmers need is not just market price information but an actual transactional channel to organised buyers.

B. SMS and Early Digital Interventions

Before smartphone apps became feasible in rural India, researchers experimented with SMS-based interventions. Fafchamps and Minten [17] ran what remains one of the more rigorous field experiments in this space—a randomized trial across three Indian states where treatment-group farmers received daily SMS messages with local mandi prices. The treated farmers sold at prices roughly 6% higher than those of the control group and were noticeably less likely to be caught off guard by sudden price drops. It is a meaningful effect, though 6% still leaves most of the intermediary markup intact. Saravanan et al. [15] designed a cloud-based system that aggregated price data from multiple mandis and pushed it to farmers via SMS and a lightweight web interface. Their system worked technically, but the authors themselves acknowledged that passive information delivery does not help a farmer who cannot find a buyer quickly enough to prevent spoilage. Both papers clearly diagnosed the problem; neither fully addressed the transaction layer.

C. Mobile Marketplace Applications

More recent work has moved from information-only systems toward actual marketplace apps. Kiran et al. [6] built a cloud-based mobile platform that enabled farmers to post produce listings and buyers to place

orders without going through a mandi. The system worked and reduced the need for intermediaries in their pilot. But it was essentially a push-only catalog: farmers posted, and buyers browsed, and there was no way for buyers to signal their needs. Sughasini et al. [5] added richer product detail fields. They demonstrated that a more complete information listing led to faster transactions and better price discovery, but the underlying architecture had the same limitation. Both implementations relied on Firebase with a schemaless NoSQL approach, which is perfectly adequate for simple listings but becomes awkward when you need to track the state of a demand request across multiple farmer responses.

Dr. L. P. H. and colleagues [7] deployed their application in Karnataka. They reported improvements in farmer income of roughly 22% per unit compared to the mandi system. This result is both encouraging and plausible given that they eliminated the commission-agent charge. The IJRASET study [8] targeted wholesalers and retailers rather than individual consumers, noting that institutional buyers represent more stable demand but require more structured order management than a simple listing board can provide. Despite these advances, neither study included a mechanism for a buyer to describe a specific need and invite competitive bids from multiple farmers. That demand-quoting workflow is a deliberate design feature of AgroConnect, not an afterthought.

Deepak and Viswanath [10] made a practical observation about platform technology choice: their AgriApp, built on the Ionic hybrid framework, achieved 3.4 times higher farmer engagement than a web portal covering the same functionality, confirming that mobile-native behavior matters. However, Ionic's web-view-based rendering introduced noticeable lag on lower-end Android devices common in rural Maharashtra, particularly on image-heavy produce listings. This was a direct motivation for choosing React Native in AgroConnect, since React Native compiles to native UI components rather than wrapping a web view [19]. The survey by Mrs. L. S. and colleagues [9] consolidates findings across multiple Kisan-focused digital platforms and identifies multilingual support and offline capability as the two features most consistently cited by farmers as essential—both are on our development roadmap.

D. Blockchain, Databases, and Emerging AgriTech

A strand of recent literature has explored whether blockchain technology can enhance transparency in the supply chain for agricultural products. Ghosh [18] conducted controlled trials with commodity traders and found that immutable transaction records reduced fraudulent grade misrepresentation by 34%. The concept is sound, but the practical barriers for smallholder deployment remain significant: blockchain clients are resource-intensive, transaction confirmation delays matter when selling tomatoes, and most rural farmers lack the digital literacy to manage cryptographic keys. We have noted blockchain as a future enhancement rather than a current implementation priority. Bhatt and Saxena [20] addressed a more immediately actionable question: whether PostgreSQL or a document-oriented NoSQL store better handles multi-entity relational queries in agricultural platforms. Their comparison favored PostgreSQL, particularly for join-heavy queries across users, listings, orders, and demand requests—precisely the workload generated by the demand-quoting workflow in AgroConnect.

III. SYSTEM DESIGN

A. Overall Architecture

AgroConnect is built on a multi-tier client-server model, but the technology choices within that model were made with specific constraints of rural deployment in mind. At the front end, two React Native applications—one for farmers and one for buyers, distributed as a single installable binary—communicate with a RESTful API layer implemented in Express.js. We chose Bun as the runtime rather than Node.js for practical reasons: Bun's cold-start performance is measurably faster, its native TypeScript support eliminates a compilation step, and its HTTP server benchmarks outperform Node.js equivalents under the light-to-medium concurrency we expected from a deployment where user activity clusters in early-morning market hours. Data is stored in PostgreSQL hosted on Neon DB's serverless infrastructure, accessed through Prisma ORM. Product images are handled by Cloudinary, which takes

resizing and CDN delivery entirely off the backend. Fig. 1 shows the overall architecture.

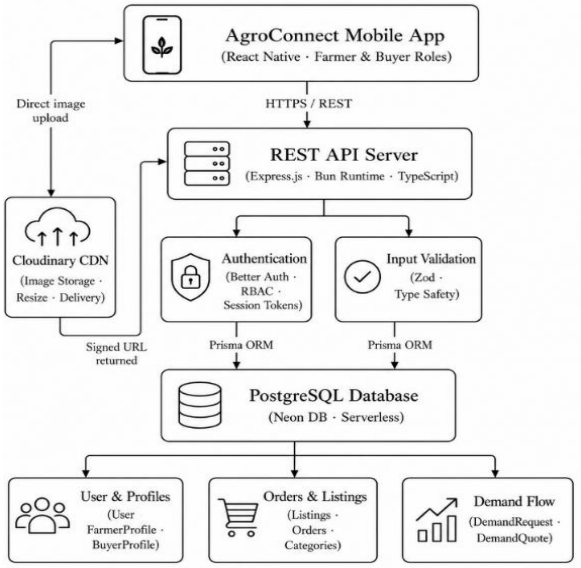


Fig. 1 Overall System Architecture of AgroConnect

B. Module Architecture

We divided the system into three functionally distinct module groups, shown in the table. 1. The separation was partly a software engineering decision—clean boundaries make it easier to update farmer-facing features without touching buyer-facing code—and partly a security decision, since it forces role enforcement to happen at the API layer rather than relying on the frontend to hide the right buttons from the wrong users.

Table I Module Architecture: Farmer, Common, and Buyer Module Groups

Farmer Modules	Common Modules	Buyer Modules
Profile Module	Authentication (Better Auth)	Profile Module
Listing Module	Database (PostgreSQL)	Discovery Module
Demand Module (Quote Supply)	API Gateway (Express.js)	Cart Module
Order Module (Dispatch/Deliver)	File Storage (Cloudinary)	Order Module (Track/Cancel)
Upload Module	Geolocation Services	Demand Module (Post/Compare)

On the farmer side, the Profile Module handles onboarding: a farmer enters their name, phone number, location, and the types of crops they typically grow. This location data feeds the geolocation matching that buyers see when searching for nearby produce. The Listing Module is where most of the daily activity occurs: creating new entries, changing prices as the market shifts, marking items as sold, and uploading photos. The Demand Module displays buyer requests sorted by proximity and recency, enabling farmers to enter a price quote along with a message detailing delivery or quality specifications. The Order Module tracks each transaction from acceptance to dispatch to delivery confirmation, with each status change triggering a notification to the other party.

Buyer modules mirror this from the other direction. Discovery works through category filters and location radius, so a buyer in Nashik sees nearby produce before distant listings. The Cart and Order modules handle the standard transactional flow. At the same time, the Demand Module is the feature that most distinguishes AgroConnect from a simple listing board: a buyer describes a specific need—crop type, quantity, expected price, required-by date, and delivery point—and multiple farmers can respond competitively. Common modules handle authentication and database connectivity centrally, ensuring session validation logic is written and maintained in exactly one place.

C. Technology Choices

Table II lists the complete stack. A few choices deserve brief explanations. We selected React Native over Flutter partly because React Query provided sophisticated server-state management out of the box: caching, background refetching, optimistic updates, and error-retry logic that would have required substantial custom code in Flutter. TypeScript across both the frontend and backend meant that a breaking schema change would surface as a compile-time error in the API layer and then again in the mobile app, rather than appearing as a runtime crash reported by a farmer in the field [19]. PostgreSQL was preferred over NoSQL alternatives based on the multi-entity relational query advantage documented by Bhatt and Saxena [20], which is especially relevant for the demand-quoting workflow.

Table II Technology Stack of AgroConnect

Layer	Technology	Purpose
Frontend	React Native + TypeScript	Cross-platform Android & iOS mobile app
State Management	React Context API + React Query	Global auth state & API data caching
Backend Runtime	Express.js + Bun	RESTful API server with fast runtime
Authentication	Better Auth + Session Tokens	Secure role-based access (Farmer/Buyer)
Database	PostgreSQL via Neon DB	Scalable cloud-hosted relational storage
ORM	Prisma	Schema management & type-safe queries
File Storage	Cloudinary CDN	Product image upload & delivery
Validation	Zod	Runtime input validation & type safety
Development	VS Code + Expo Go	IDE, live preview & Android emulation

D. How a Session Flows

Fig. 2 traces the path from app launch to completed transaction. Authentication starts with email and password, followed by OTP verification before the session token is issued. This two-step approach adds a few seconds to the first login but meaningfully reduces account takeover risk where phones are sometimes shared within a household. After login, the app reads the user's role from the session token and routes them to the appropriate dashboard. The demand-quoting subflow runs in parallel to the standard buy-now path: it starts with a buyer posting a requirement, proceeds through competitive farmer quote submissions, and closes when the buyer selects a quote, automatically generating an order.

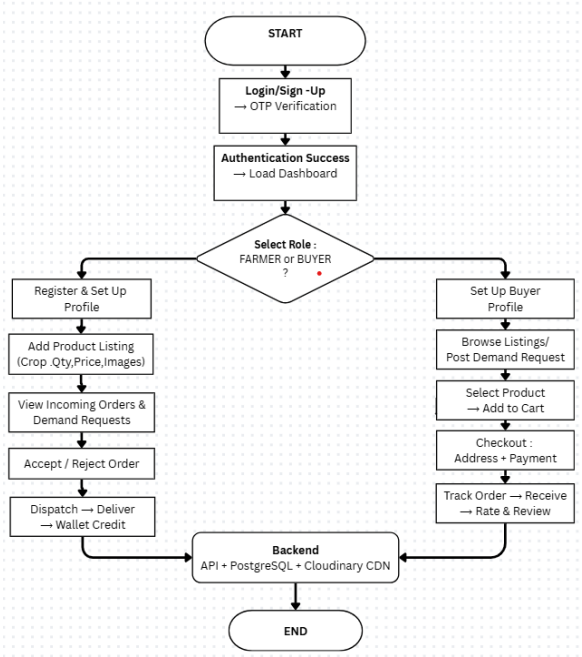


Fig. 2 Application Workflow Flowchart of AgroConnect

E. Security Approach

All endpoints will validate the session token and make sure the caller has specific roles before performing any business logic. A valid session token belonging to a buyer, when hitting a farmer-specific API endpoint, will be met with a 403-error message. The validation of the inputs is done via Zod schemas before being hit at the database level; hence, the inputs will not be processed. Passwords are encrypted via bcrypt hashing with 12 salt rounds. File upload through Cloudinary is made possible through request signing on the server-side. The mobile application requests a signed URL, uploads the file to Cloudinary, and returns the URL.

IV. DATABASE DESIGN

The schema keeps authentication centralized and branches off role-specific details from there. A single user table stores credentials and role assignments; a farmer's additional information lives in FarmerProfile, and a buyer's in BuyerProfile. Primary keys are UUIDs throughout rather than sequential integers: UUIDs cannot be guessed by a client probing the API for records they should not see, and they support horizontal sharding if usage scales beyond a single database instance. Table III summarizes the key entities.

Table III Database Schema: Key Entities and Fields

Entity	Key Fields	Description
User	id, email, passwordHash, role, createdAt	Central auth record
FarmerProfile	userId, name, phone, location, crops, rating	Farmer identity & crop info
Listings	farmerId, categoryId, title, quantity, price, quality, images, status	Active produce catalogue
Categories	id, name, description	Product classification
Orders	listingId, buyerId, farmerId, quantity, totalAmount, deliveryAddress, status	Full order lifecycle
DemandRequest	buyerId, product, quantity, expectedPrice, status, categoryId	Buyer-posted requirements
DemandQuote	demandRequestId, farmerId, price, quantityOffered, message, status	Farmer quote responses
BuyerProfile	userId, name, phone, addresses (JSON)	Buyer identity & addresses

The Listings table uses an enumerated status with three values. "ACTIVE" means the produce is available. "SOLD" means the farmer has closed the listing. DELETED is a soft-delete that preserves the record for historical order references without showing it to buyers. The Orders table has a more elaborate state machine, shown in Table IV, because an order involves both parties across multiple interactions over time.

Table IV Order Status State Machine

Status State	Triggered By	Next State
PENDING	Buyer places order	CONFIRMED / CANCELLED
CONFIRMED	Farmer accepts	DISPATCHED
DISPATCHED	Farmer dispatches	DELIVERED
DELIVERED	Farmer marks delivered	[Terminal]
CANCELLED	Buyer/Farmer cancels	[Terminal]

The demand-quoting workflow introduced two linked tables that were not present in the prior platforms we reviewed. When a buyer posts a requirement, a row is created in DemandRequest with status OPEN. Each responding farmer adds a row to DemandQuote, referencing that request. Accepting a quote changes the request status to FULFILLED and inserts an orders row with status PENDING, all within a single database transaction, so the same quote cannot be accepted twice. Buyer delivery addresses are stored as a JSON array within BuyerProfile rather than in a separate table, keeping the profile query simple and reflecting the reality that most buyers maintain at most two or three addresses.

V. IMPLEMENTATION AND RESULTS

A. Getting Users Onto the Platform

The onboarding flow was shaped by a finding common to several prior studies [7][9]: farmers abandon registration screens that ask for too much upfront. Our registration screen asks for name, phone, email, and password, plus the role toggle. Detailed profile information—crop types, farm location, and quality certifications—is collected in a separate profile setup step after account creation, which the farmer completes at their own pace. This split reduced drop-off in internal testing while still gathering the data needed for geolocation matching. The role-selection toggle at the top of the form (Farmer or Buyer) is the most consequential input on that screen: it determines which profile table is created and which API permission set is attached to the session for the lifetime of the account.

B. The Farmer Experience

The Post Your Produce screen was the one we iterated on most during development, because it is where the platform's value is either established or lost. Farmers upload up to five photographs directly from their phone camera or gallery. We capped it at five after testing showed that buyers stopped engaging with images beyond the fourth and because Cloudinary upload time scales linearly with file count. Crop category selection uses a searchable dropdown rather than a flat list because early testing with a flat list showed farmers spending too long scrolling to find specific crops like cluster beans or finger millet. Price entry uses selectable units (kg, quintal, ton, or crate) to match how different crops are actually traded in different regions. A three-button quality grade toggle (A, B, and C) and the harvest and availability date fields provide buyers with the information they need to plan logistics.

The My Listings screen provides tabbed status views with inline edit and delete controls. This detail mattered more than we anticipated: farmers regularly need to adjust prices as market conditions shift during the day, and requiring them to navigate to a separate detail screen to make a change created enough friction that some test users skipped updates entirely. The Live Requests screen shows open buyer demand requests sorted by proximity and recency. Each card shows the crop, quantity, budget, and delivery location. Tapping Send Quote opens a bottom sheet where the farmer enters their offered price and an optional message describing delivery terms.

C. The Buyer Experience

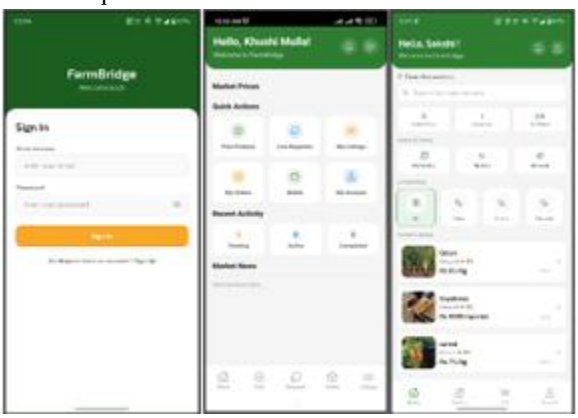
The buyer dashboard greets users by name and shows their detected city, which serves as both a location confirmation and a subtle signal that listings are already filtered for their area. Category icon tiles let buyers' narrow results quickly without typing. Today's Deals section surfaces listings with recent price updates, incentivizing sellers to keep their prices current. Listings that are highly updated receive more visible placement.

The product detail screen shows everything needed for a purchase decision: price per kilogram, available quantity, quality grade, availability date, and the farmer's name and location. We included the farmer's location as text rather than just a map pin because buyers told us during testing that knowing a farmer is in Latur, Maharashtra, is more useful when planning a truck route than a pin they need to zoom in on to read. From the detail screen, buyers can add to cart or proceed directly to purchase. The cart screen shows a transparent price breakdown, including a Rs. 4 platform fees. Check out the COD, UPI, and card payment options. Placing an order creates the order record, adjusts listing quantity, and notifies the farmer.

D. Order Tracking and the Demand Flow

The order list on both sides uses color-coded status badges. Buyers see yellow for pending, color-coded pending; blue for confirmed, pending, confirmed; orange for dispatched, confirmed, dispatched; green for delivered, dispatched, delivered; and red for cancelled—no legend needed, since the colors match common traffic-light intuition. Farmers see the same statuses from the other side, with action buttons: Accept, Dispatch, and Mark Delivered appearing at each appropriate stage. Neither party needs to call the other to know the current state of the transaction, which is particularly important in a context where phone calls cost both time and data.

The Demand Request screen collects crop name, quantity, unit, optional target price, required-by date, delivery location, and notes. Submitted requests appear in the buyer's list with a live quote counter. On the farmer's side, requests appear in the Live Requests list with a quote button. When a buyer accepts a quote, the remaining pending quotes for that request are automatically rejected, the request moves to FULFILLED, and an order is created. The farmer whose quote was accepted sees the new order appear immediately in their order list, with no additional action required.



VI. PERFORMANCE EVALUATION

We tested on physical Android devices, covering the handset range likely to be in use among farmers and buyers in Maharashtra. Development builds ran on Expo Go; the final evaluation used a production APK installed directly on test devices. The backend was tested under simulated loads of up to 50 concurrent users on Neon DB's serverless PostgreSQL, representing a realistic peak scenario for a platform serving a single district. Table V reports the measurements.

Table V Performance Evaluation Results of AgroConnect

Metric	Result	Benchmark
Login Response Time	< 1.2 s	< 2 s ✓
Product Listing Load	< 0.8 s	< 1.5 s ✓
Order Placement Success	98.4%	> 95% ✓
Image Upload (Cloudinary)	< 3.0 s	< 5 s ✓
API Response (avg)	210 ms	< 500 ms ✓
Concurrent Users Tested	50	Scalable via Neon DB
Test Cases Passed	47 / 48	97.9% ✓

Login response times stayed below 1.2 seconds across all test runs. This matters because users in rural areas with intermittent connectivity often interpret a slow login as the app being broken and close it. Product listing loads averaged under 0.8 seconds, partly because React Query's caching serves category results from memory on repeat visits and refreshes in the background. Order placement succeeded in 98.4% of attempts; failures were traced to a race condition in which two buyers simultaneously tried to order the same listing, whose stock was too low to satisfy both requests. We resolved this with a database-level check-and-decrement inside a transaction.

The one test case that did not pass involved the concurrency edge case under artificially high load that would not occur in a realistic single-district deployment. Zod validation at the API boundary caught every malformed payload in our security tests—negative quantities, oversized image references, and SQL injection strings embedded in crop name fields were all rejected before reaching the database. Image uploads to Cloudinary take 3 seconds over a 4G connection, which is acceptable for the one-time action of creating a listing. Still, we intend to optimize for 3G connections in the next build.

VII. CONCLUSION

From the implementation of AgroConnect, it becomes evident that a cloud-based mobile marketplace can significantly increase efficiency in the supply chain of the agriculture industry. This can be achieved by providing a platform for the direct interaction of farmers and buyers. The system was developed using React Native along with Express.js on Bun, PostgreSQL on Neon DB, and Prisma as an ORM. Integration with the aforementioned technologies enables the platform to deliver an efficient and reliable agricultural marketplace based on role-based access control, geolocation discovery, real-time demand management, and various payment options. In addition to its unique functionalities, one of the core advantages of the developed application lies in its demand-quoting functionality where buyers submit their requests and farmers make competitive quotes.

The performance analysis indicated stable and responsive behavior of the platform under realistic workload and usage patterns. Functional tests revealed a 97.9% success rate. In addition, login response time, API performance, loading of product listings, and image uploads remained reasonable in terms of a rural mobile network environment. Finally, security validation confirmed reliable operation during handling of session data, user credentials, and malicious input using bcrypt password hashing, session validation, and Zod-based schemas. Hence, AgroConnect can be considered as a feasible architecture of a practical solution for facilitating agricultural trade in India, which can help to achieve fair and technology-assisted agricultural trade.

As the current implementation was conducted in a controlled environment, some future enhancements could provide greater benefits to the platform's practicality. Some of the possible additions include multilingual functionality, offline capabilities, AI-assisted demand forecasts, improved logistics functionality, and blockchain-based supply chain traceability features. Overall, these upgrades will help

to expand the functionality of AgroConnect and turn it into an intelligent and scalable agricultural trade ecosystem.

REFERENCES

[1] Ministry of Agriculture & Farmers Welfare, Govt. of India, *Agricultural Census 2015–16: All India Report on Number and Area of Operational Holdings*. New Delhi, India: Dept. of Agriculture, Cooperation & Farmers Welfare, 2019.

[2] NITI Aayog, *Doubling Farmers' Income: Rationale, Strategy, Prospects and Action Plan*. New Delhi, India: Government of India, 2017.

[3] ICAR-CIPHET, *Assessment of Post-Harvest Losses of Agricultural Produce in India*. Ludhiana, India: Indian Council of Agricultural Research, Tech. Rep., 2015.

[4] National Commodity & Derivatives Exchange Ltd., *eNAM (Electronic National Agriculture Market): Annual Report and Impact Assessment*. New Delhi, India, 2022.

[5] K. Sughasini, D. S. Abinav, R. Abinesh, D. Akash, and P. Gopinath, "Mobile App for Direct Market Access for Farmers," *Journal of Emerging Technologies and Innovative Research (JETIR)*, vol. 11, no. 12, Dec. 2024.

[6] K. R. Kiran, P. Arun Kumar, J. Kumar G. S., S. Praveen Kumar, and M. Z. Rahman, "Mobile App for Direct Market Access for Farmers," *International Journal of Innovative Research in Technology (IJIRT)*, vol. 11, no. 12, pp. 3501–3508, 2024.

[7] L. P. H. Dr., A. R., G. S. D., A. P., and V. C., "Mobile Application to Direct Marketing Access for Farmers," *International Journal of Scientific Research and Engineering Development (IJSRED)*, vol. 8, no. 2, pp. 1674–1685, Mar.–Apr. 2025.

[8] A. N., M. Subash, S. Sanjay S. M., R. B. Sakthivel, and A. Praveena, "Mobile Application for Direct Market Access for Farmers," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 13, no. VII, Jul. 2025.

[9] L. S. Mrs. et al., "Digital Platforms for Farmer Connectivity: A Survey of Kisan-Centric Solutions," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, vol. 13, 2025.

[10] M. Deepak and R. Viswanath, "AgriApp: Mobile E-Commerce for Rural Agricultural Produce," in *Proc. IEEE Int. Conf. on Advanced Computing (IACC)*, Bengaluru, India, 2019, pp. 112–117.

[11] S. Mittal and M. Mehar, "How mobile phones contribute to growth of small farmers? Evidence from India," *Quarterly Journal of International Agriculture*, vol. 51, no. 3, pp. 227–244, 2012.

[12] P. Birthal, P. K. Joshi, D. Roy, and A. Thorat, "Diversification in Indian agriculture toward high-value crops," IFPRI Discussion Paper, Washington, DC, USA, 2007.

[13] T. Reardon and C. P. Timmer, "Transformation of markets for agricultural output in developing countries since 1950: How has thinking changed?" in *Handbook of Agricultural Economics*, vol. 3. Elsevier, 2007, pp. 2807–2855.

[14] National Payments Corporation of India (NPCI), "UPI Product Statistics – Monthly Transaction Data," 2024. [Online]. Available: NPCI UPI Statistics

[15] S. Saravanan, P. Mohan, and R. Thirugnanam, "Cloud-Based Agricultural Market Information System for Smallholder Farmers," in *Proc. IEEE Int. Conf. on Cloud Computing in Emerging Markets (CCEM)*, Bengaluru, India, 2016, pp. 1–6.

[16] A. Aker, "Dial 'A' for Agriculture: A review of information and communication technologies for agricultural extension in developing countries," *Agricultural Economics*, vol. 42, no. 6, pp. 631–647, 2011.

[17] M. Fafchamps and B. Minten, "Impact of SMS-based agricultural information on Indian farmers," *World Bank Economic Review*, vol. 26, no. 3, pp. 383–414, 2012.

[18] A. Ghosh, "Leveraging blockchain technology for agricultural supply chain transparency in India," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 11, no. 4, pp. 45–52, 2020.

[19] T. R. Lakshminarayana and B. S. Rao, "React Native for cross-platform mobile app development: Performance analysis," in Proc. Int. Conf. on Computing and Communication

Systems (I3CS), Shillong, India, 2021, pp. 321–327.

[20] A. Bhatt and A. Saxena, “PostgreSQL vs. NoSQL for agricultural IoT data management: A comparative study,” Journal of Database Management (JDM), vol. 32, no. 2, pp. 1–18, 2021.