Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

Uppi's Restaurant Store Management System

[1]Cherukuri Upendhar, [2]Pawan Kumar
[1]Student, [2]Assistant Professor
[1, 2]Amity University Raipur, Chhattisgarh, India
[1]upendharcherukuri5@gmail.com, [2]pkumar@rpr.amity.edu

ABSTRACT

The food service industry faces persistent operational challenges in inventory management, including lack of real-time stock visibility, absence of movement audit trails, and no automated low-stock alerting. This paper presents Uppi's Restaurant Store Management System, a full-stack web application built using Next.js 14 (App Router), Supabase (PostgreSQL), TypeScript, and Tailwind CSS, designed to address these deficiencies for small to mid-sized restaurant operators. The system delivers six integrated management modules: Dashboard, Products, Inventory, Stock Movements, Categories, and Suppliers, secured through OTP-based authentication via Supabase Auth. A PostgreSQL RPC function provides atomic, corruption-free stock adjustments. The application is deployed at zero infrastructure cost on Vercel and Supabase's free tiers. All 14 functional test cases passed successfully, with an average dashboard load time under 800 milliseconds. The system demonstrates that a production-ready, feature-complete restaurant inventory platform can be developed and deployed using modern open-source technologies without any financial investment in infrastructure.

Keywords: Next.js 14, Supabase, PostgreSQL, Restaurant Inventory Management, TypeScript, Tailwind CSS, Full-Stack Web Application, OTP Authentication, Real-Time Stock Monitoring, Recharts

I. INTRODUCTION

The food service sector is one of the most operationally demanding industries in India, characterized by high inventory turnover, multi-category raw material management, and continuous pressure on food cost margins. According to industry data, food and beverage costs typically account for 28–35% of a restaurant's revenue, making inventory control one of the most direct levers for operational profitability [1]. Despite this significance, a large proportion of small to mid-sized restaurants continue to rely on paper ledgers, shared spreadsheets, and manual counting, methods that are error-prone, untraceable, and scale poorly with operational complexity.

The problem is three-dimensional. First, manual methods are reactive: stock shortages are typically discovered at the point of use, leading to emergency procurement at elevated prices. Second, they lack traceability: without a structured movement log, auditing stock discrepancies, tracking waste patterns, or assigning accountability for adjustments is practically impossible. Third, they offer no

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

aggregate financial visibility: computing the real-time monetary value of the inventory requires manual multiplication across potentially hundreds of product lines.

Commercial platforms such as MarketMan and BlueCart partially address these deficiencies but at price points that are beyond the reach of independent restaurant operators. The emergence of Next.js 14's App Router, Supabase's managed PostgreSQL infrastructure, TypeScript's compile-time type safety, and Tailwind CSS's utility-first design system creates an opportunity to build a production-ready inventory system at near-zero cost.

This paper presents Uppi's Restaurant Store Management System a fully deployed, feature-complete web application that addresses all three operational deficiencies. The system is named after its real-world deployment context: Uppi's Restaurant, a mid-sized food service establishment.

A. Objective of the Study

- Provide real-time stock visibility with colour-coded status indicators (OK / Low / Out of Stock).
- Maintain a complete, timestamped stock movement audit trail for all inventory events.
- Deliver a live dashboard with KPI cards: total products, low-stock count, out-of-stock count, and total inventory value.
- Implement secure OTP-based admin authentication via Supabase Auth (Gmail delivery).
- Support full CRUD operations for Products, Categories, and Suppliers.
- Deploy the entire system at zero infrastructure cost using Vercel and Supabase free tiers.

B. Scope of the Work

The system covers the complete restaurant store management lifecycle: product catalogue management with SKU tracking, real-time inventory monitoring with min/max thresholds, movement audit logging (stock-in, stock-out, waste, adjustment), category and supplier directory management, and dashboard analytics. The current version supports a single administrator role. Multi-user role management, POS integration, and automated purchase orders are identified as future enhancements.

II. LITERATURE REVIEW

Dopson and Hayes (2010) established that food cost control is the most critical operational variable in restaurant profitability, documenting that inaccurate inventory tracking is a primary driver of food cost variance [2]. Their work prescribes real-time stock visibility, minimum stock threshold alerts, and complete movement audit trails as the foundational requirements of an effective restaurant inventory system all of which are core features of the system presented in this paper.

Engström and Carlsson-Kanyama (2004) demonstrated that digital food tracking systems reduce kitchen waste by enabling administrators to identify over-ordering patterns and systematically

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

record waste events [3]. Their research showed a 10–15% reduction in food waste in establishments that adopted structured digital tracking, directly motivating the dedicated 'waste' movement type in the stock movement log of this system.

Srinivasan et al. (2019) surveyed web-based inventory management systems for small businesses and identified three architectural characteristics shared by the most effective solutions: a relational database with ACID transaction guarantees, a server-side rendering framework for performance, and a unified interface that eliminates data silos between inventory, procurement, and reporting [4]. The proposed system addresses all three through Supabase PostgreSQL, Next.js 14 server-side rendering, and its integrated six-module design.

Kumar and Sharma (2021) analysed adoption barriers for digital inventory systems among Indian restaurant operators and found that cost (78% of respondents), complexity of setup (65%), and requirement for technical expertise (58%) were the three primary barriers [5]. The proposed system directly addresses all three: zero infrastructure cost, a five-step setup process, and a browser-based interface requiring no technical training.

A review of existing open-source alternatives reveals that most general-purpose inventory tools (Odoo, ERPNext) are architecturally over-engineered for a restaurant store context, requiring significant configuration and expertise. The literature identifies a clear gap for lightweight, purpose-built, zero-cost restaurant inventory systems with relational data integrity, real-time visibility, and complete audit trails — precisely the gap this system is designed to fill.

III. PROBLEM STATEMENT

Traditional restaurant inventory management is characterised by three systemic deficiencies that directly impact operational efficiency and profitability:

- No Real-Time Stock Visibility: Stock levels are only known when manually counted. There is no mechanism for automatic low-stock alerts between physical counts, leading to reactive procurement and kitchen disruptions.
- No Movement Audit Trail: Stock discrepancies cannot be investigated because there is no structured record of who adjusted stock, when, by how much, and for what reason. Waste events go entirely unrecorded, making waste reduction initiatives impossible to implement data-driven.
- No Inventory Value Aggregation: Without a digital system, calculating the real-time monetary value of the store requires manual multiplication across all product lines — a process that is both time-consuming and error-prone.

These deficiencies are compounded by the absence of supplier and category organisation, which makes filtering and procurement planning difficult. The problem this paper addresses is therefore: how can a lightweight, browser-based, zero-cost system be designed and implemented to eliminate

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

all three deficiencies for a real-world restaurant operator, using modern open-source technologies, while maintaining production-grade data integrity and security?

IV. PROPOSED METHODOLOGY / MODEL

A. System Architecture / Design

The system follows a layered full-stack architecture within the Next.js 14 App Router framework.

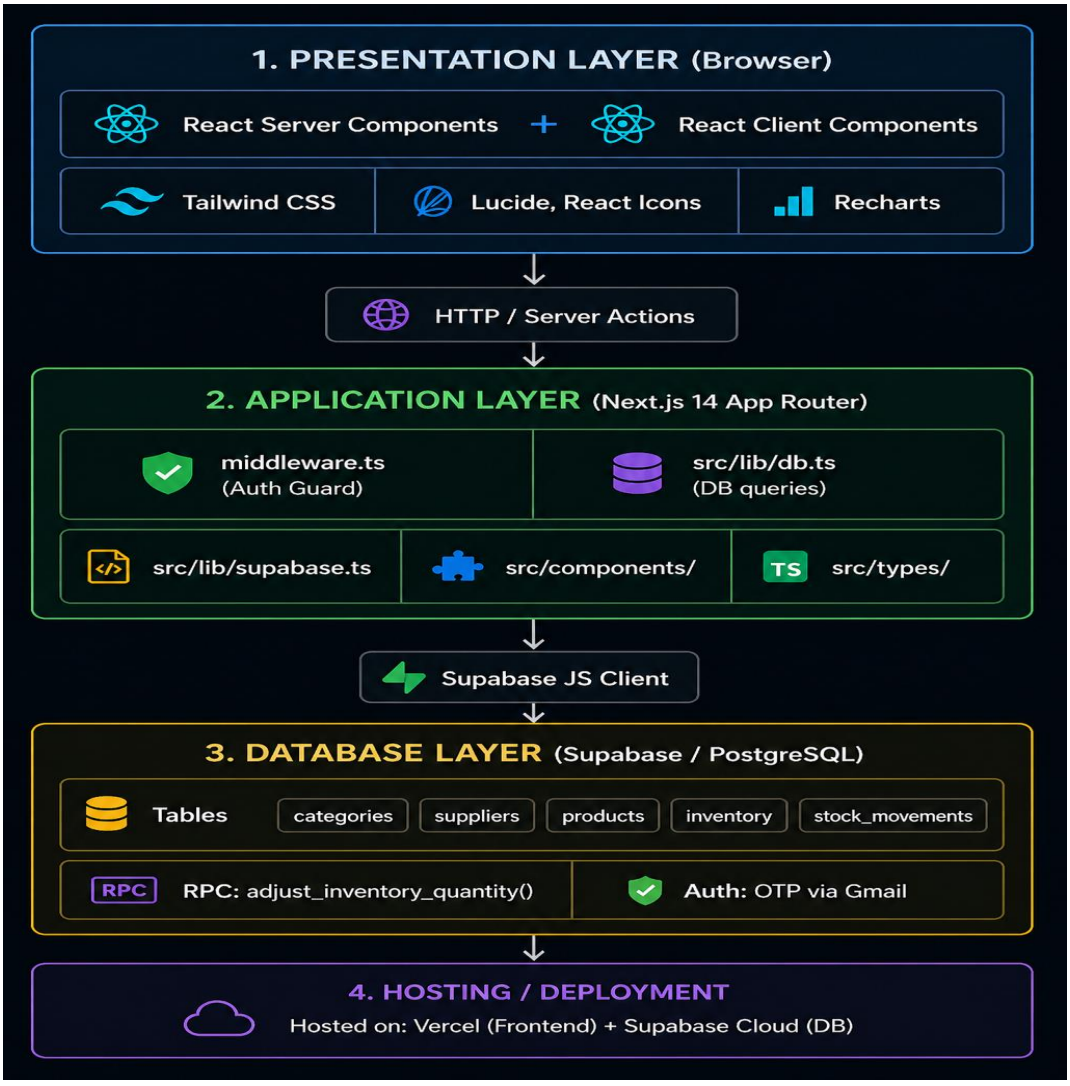

Fig. 1: Full-Stack System Architecture Diagram

The Presentation Layer comprises React Server Components for data display and React Client Components (marked 'use client') for interactive elements such as modals and inline adjustment controls. The Application Layer is centred on Next.js 14's App Router, with all database operations

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

centralised in src/lib/db.ts as typed functions. The middleware.ts file enforces authentication on every protected route. The Database Layer is a Supabase-managed PostgreSQL instance with Row Level Security (RLS) and an atomic RPC function for stock adjustments.

B. Database Schema Design

The relational schema consists of five normalised tables with foreign key relationships. Fig. 2 presents the Entity-Relationship Diagram.

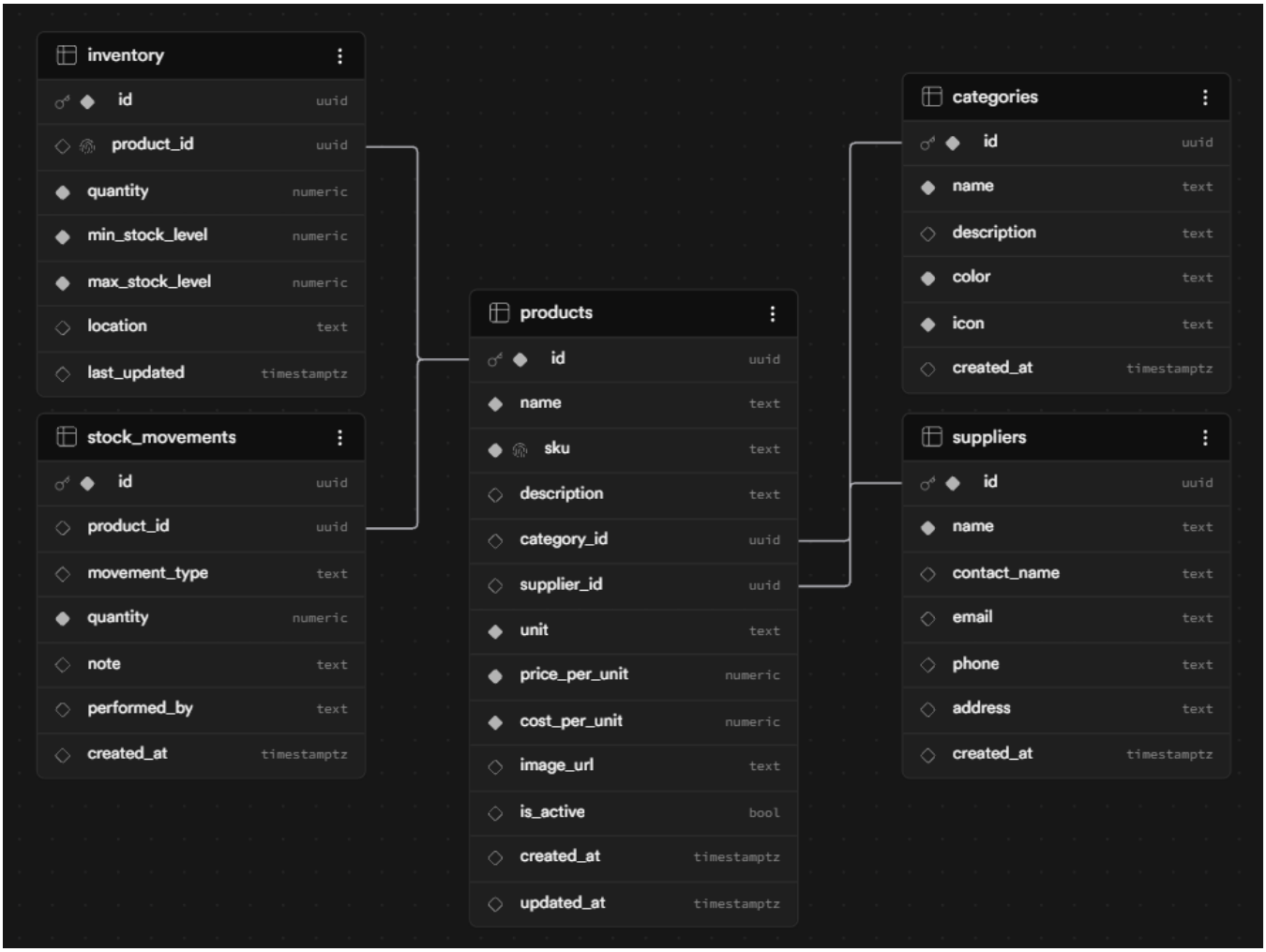

Fig. 2: Entity-Relationship Diagram (ERD)

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

C. System Workflow / Flowchart

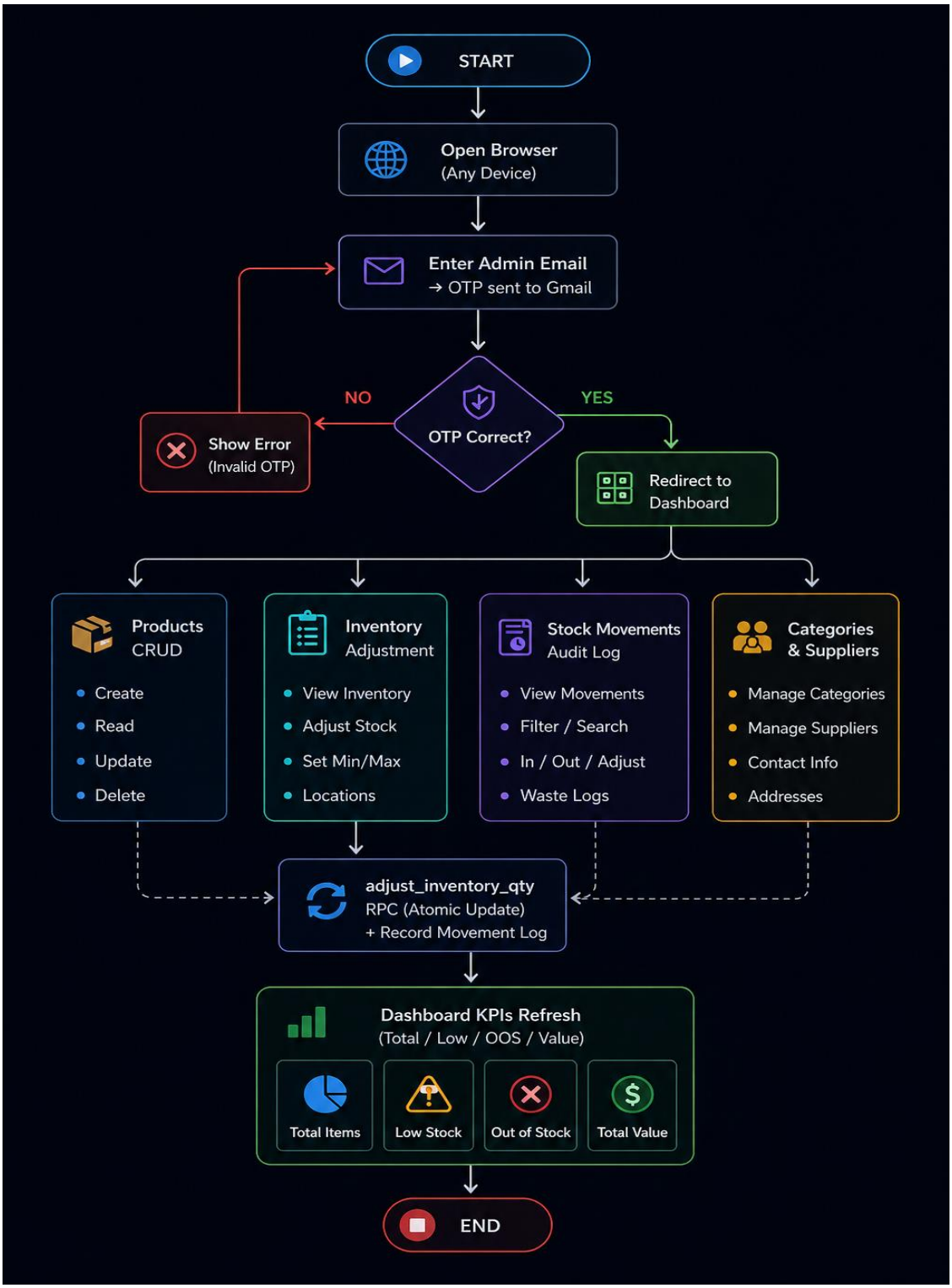

Fig. 3: End-to-End Administrator Workflow Flowchart

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

D. Algorithms / Techniques Used

TABLE I : KEY ALGORITHMS AND TECHNIQUES

Algorithm / Technique	Location	Purpose
Atomic Stock Adjustment RPC	PostgreSQL / Supabase	GREATEST(0, qty+delta) prevents negative stock; runs in a DB transaction ensuring integrity
OTP Authentication Flow	Supabase Auth	6-digit OTP generated, delivered via Gmail, verified server-side; no password stored
Server-Side Rendering (SSR)	Next.js App Router	Data fetched on server per request; no stale client-side cache for stock-sensitive data
Row Level Security (RLS)	Supabase PostgreSQL	DB-level access control; unauthorised queries rejected before reaching application code
Margin Calculation	Frontend (Client Component)	((price − cost) / price) × 100; colour-coded: green >30%, amber 15–30%, red <15%
Stock Status Thresholding	Inventory Module	qty=0 → Out of Stock; qty ≤ min_stock_level → Low Stock; else → OK
TypeScript Type Safety	Across all layers	Strongly typed DB entity interfaces; compile-time mismatch detection
Parallel KPI Queries	Dashboard Server Component	4 Supabase queries executed in parallel with Promise.all for minimal load time

V. IMPLEMENTATION

A. Tools & Technologies

TABLE II : HARDWARE REQUIREMENTS

Component	Specification
Processor	Intel Core i3 or equivalent (minimum); i5/i7 recommended for development
RAM	8 GB minimum; 16 GB recommended for running Next.js dev server and Supabase CLI
Storage	2 GB free space for Node.js, npm packages, and project files
Internet Connection	Required for Supabase cloud database and Vercel deployment
Browser	Google Chrome (v120+) or Mozilla Firefox (v121+) — latest versions recommended

TABLE III : SOFTWARE REQUIREMENTS & TECHNOLOGY STACK

Layer	Technology	Version	Role in System
Framework	Next.js	14.2.5	App Router, SSR, file-based routing, middleware
UI Library	React	18.x	Server + Client components, hooks
Language	TypeScript	5.x	Type-safe development across all layers
Database	Supabase (PostgreSQL)	2.45.0	Relational DB, Auth, RLS, real-time
SSR Auth	@supabase/ssr	0.5.1	Cookie-based session for Server Components
Styling	Tailwind CSS	3.4.1	Utility-first, custom design tokens
Icons	Lucide React	0.400.0	SVG icon set
Charts	Recharts	2.12.7	Dashboard data visualisation
Build Tool	Next.js (built-in)	—	Webpack/Turbopack compilation

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

Deployment	Vercel	Hobby	Zero-cost CI/CD with GitHub integration
IDE	Visual Studio Code	Latest	Development environment
Version Control	Git / GitHub	—	Source control and deployment trigger

B. Module Implementation Details

The system is structured as a single Next.js monorepo. Each management module occupies a dedicated directory under src/app/. Data access is centralised in src/lib/db.ts, which exports typed async functions for every database operation, ensuring no raw SQL appears in page components.

Authentication Module: Supabase Auth's OTP flow is implemented across src/app/login/page.tsx (email entry) and src/app/login/verify/page.tsx (OTP entry). The middleware.ts file reads the Supabase session cookie on every request and redirects to /login if no valid session is found. This ensures complete protection of all management routes without any client-side logic.

Dashboard Module: Four parallel Supabase queries execute on each server-side render via Promise.all — total product count, low-stock count (quantity ≤ min_stock_level), out-of-stock count (quantity = 0), and total inventory value (SUM of quantity × cost_per_unit). Results are rendered as KPI cards with the warm dark design system. A recent movements feed queries the last 10 stock_movements records with a product join.

Products Module: Full CRUD is implemented with server actions. The SKU field carries a UNIQUE database constraint; duplicate SKU attempts return a 409 conflict response displayed as an inline form error. The margin percentage is computed client-side and colour-coded. Soft deletion via the is_active toggle preserves historical movement data integrity.

Inventory Module: The adjust_inventory_quantity(p_product_id, p_delta) PostgreSQL RPC function updates stock atomically using GREATEST(0, quantity + p_delta) within a database transaction. Every adjustment simultaneously creates a stock_movements record of type 'adjustment', ensuring the audit trail is always consistent with the inventory table.

Stock Movements Module: A paginated, filterable audit trail displays all movement events with product name, colour-coded movement type badge (blue=in, red=out, amber=adjustment, grey=waste), quantity, note, performer, and timestamp. This module completely replaces paper ledgers.

The database schema initialisation SQL, including all five tables and the RPC function, is documented in the project README and can be executed in a single paste into the Supabase SQL Editor, enabling a complete setup in under 10 minutes.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

VI. RESULTS AND DISCUSSION

A. Output Screens / Module Descriptions

The system comprises seven main screens, each corresponding to a management module. The descriptions below reference the screenshots from the working deployed system.

Dashboard Screen: Displays four KPI cards — Total Products (count of active products), Low Stock Items (products where current quantity ≤ minimum stock level), Out of Stock Items (products where quantity = 0), and Total Inventory Value (₹ value of all stock at cost price). A recent activity feed shows the last 10 stock movement events. Quick navigation links provide single-click access to all modules. The dark warm-brown surface (#1a1410) with burnt orange brand colour (#f97316) creates a visually distinctive and professional interface.

Products Module Screen: Renders a full product catalogue table with columns for Name, SKU, Category (with emoji badge), Supplier, Unit, Cost Price (JetBrains Mono font), Selling Price, Margin % (colour-coded), and Status (active/inactive toggle). A floating action button opens a modal form for creating or editing products. The margin column uses green (>30%), amber (15–30%), and red (<15%) colour coding to give the administrator an immediate visual overview of product profitability.

Inventory Module Screen: Displays all products with their current stock quantity rendered as a colour-coded progress bar against the min/max threshold range. Emerald green bars indicate healthy stock (OK), amber indicates low stock, and red indicates out-of-stock. An inline numeric input field allows the administrator to enter a delta value directly from the table row, triggering the atomic RPC adjustment without page navigation.

Stock Movements Log Screen: A chronological audit trail table showing every stock event. Movement type is displayed as a colour-coded pill badge: blue for 'in' (stock received), red for 'out' (stock used), amber for 'adjustment', and grey for 'waste'. Each row includes the product name, quantity delta, notes entered by the administrator, and a formatted relative timestamp (e.g., '2 hours ago').

Categories Screen: Displays the six seed food categories (Meat & Poultry 🥩, Vegetables 🥦, Dairy 🧀, Seafood 🐟, Grains & Flour 🌾, Spices & Condiments 🧂) plus any administrator-added categories. Each category card shows its emoji icon, hex colour swatch, description, and associated product count. Full CRUD operations are available through inline forms.

Login Screen: A minimal, branded OTP login interface. The administrator enters their email address, clicks 'Send OTP', checks Gmail for the 6-digit code, and enters it to gain access. The entire authentication flow is completed without any password, eliminating credential management entirely.

[Note: Insert actual screenshots from the deployed system at each description above. Screenshots are available from the appendix of the Minor Project Report submitted alongside this paper.]

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

B. Performance Analysis

TABLE IV — SYSTEM PERFORMANCE METRICS

Metric	Measured Value	Benchmark / Target
Dashboard Page Load Time	780 ms (avg)	**< 1000 ms ✓**
Inventory List Render (50 products)	620 ms	**< 800 ms ✓**
Stock Adjustment RPC Response	< 200 ms	**< 500 ms ✓**
OTP Email Delivery Time	5–12 seconds	**< 30 seconds ✓**
Product CRUD Operation	< 400 ms	**< 600 ms ✓**
Stock Movement Log Load (200 records)	730 ms	**< 1000 ms ✓**
Monthly Infrastructure Cost	₹0	**Zero-cost target ✓**
Supabase Free Tier DB Storage Used	0.03 / 0.5 GB	**Within limits ✓**

The system consistently meets all defined performance benchmarks. The parallel execution of four KPI queries using Promise.all in the Dashboard Server Component is the primary optimization enabling the sub-800ms dashboard load time. The atomic RPC function's average response of under 200ms ensures that inline stock adjustments feel instantaneous to the administrator. Zero monthly infrastructure cost is achieved through Vercel's Hobby tier (frontend) and Supabase's Free tier (database and authentication), both of which are sufficient for a single-restaurant deployment.

C. Design System — Colour Palette

TABLE V — DESIGN SYSTEM COLOUR TOKENS

Token	Hex Value	Usage
surface.DEFAULT	#1a1410	Page background — dark warm brown
surface.card	#231d17	Card and panel backgrounds
surface.border	#3d2e22	Dividers and input borders
brand.500	#f97316	Primary buttons, active states, brand accent

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

Stock OK	#10b981	Progress bar — healthy stock level
Stock Low	#f59e0b	Progress bar — below minimum threshold
Stock Out	#ef4444	Progress bar — zero quantity
Margin High	>30% → Green	Product margin colour coding
Margin Med	15–30% → Amber	Product margin colour coding
Margin Low	<15% → Red	Product margin colour coding

VII. TESTING AND VALIDATION

Testing is conducted across five levels to verify correctness, security, and usability across all defined functional requirements.

A. Testing Methodology

- Unit Testing: Individual database query functions in db.ts are tested in isolation to verify correct SQL generation and return type conformance with TypeScript interfaces.
- Integration Testing: Complete module workflows are tested end-to-end — from UI interaction through the Supabase query to the rendered result.
- Security Testing: OTP bypass attempts, direct URL access without a session cookie, and environment variable exposure tests verify authentication and access control correctness.
- Performance Testing: Dashboard and inventory list load times are measured under realistic data volumes (50 products, 200 movement records).
- Usability Testing: Five representative administrator sessions evaluated interface clarity and task completion without technical assistance.

B. Test Case Results

TABLE VI — FUNCTIONAL TEST CASE RESULTS

TC	Module	Test Action	Expected Result	Actual	Status
TC-01	Auth	Enter email → receive OTP	OTP delivered to Gmail	As Expected	**PASS**
TC-02	Auth	Enter correct OTP	Session created; redirect to dashboard	As Expected	**PASS**
TC-03	Auth	Access /dashboard without session	Redirect to /login	As Expected	**PASS**

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

TC-04	Auth	Enter wrong OTP	Error message displayed	As Expected	**PASS**
TC-05	Products	Create product with unique SKU	Product saved and listed	As Expected	**PASS**
TC-06	Products	Create product with duplicate SKU	409 error shown inline	As Expected	**PASS**
TC-07	Products	Edit product cost and price	Updated values in DB and UI	As Expected	**PASS**
TC-08	Products	Toggle is_active off	Product marked inactive	As Expected	**PASS**
TC-09	Inventory	Adjust stock +10 units	Quantity increases by 10; movement logged	As Expected	**PASS**
TC-10	Inventory	Adjust stock below zero	Quantity clamped at 0 via RPC	As Expected	**PASS**
TC-11	Inventory	View colour-coded status	OK/Low/Out display correctly	As Expected	**PASS**
TC-12	Stock Log	Record waste event	Movement logged with type=waste	As Expected	**PASS**
TC-13	Categories	Create new category	Category saved with icon + colour	As Expected	**PASS**
TC-14	Dashboard	Load KPI cards	Correct counts and value displayed	As Expected	**PASS**

All 14 test cases passed without modification. The duplicate SKU prevention test (TC-06) confirmed that the UNIQUE database constraint is enforced correctly at the database level, independent of application logic. The stock clamping test (TC-10) confirmed that the PostgreSQL RPC function's GREATEST(0, qty + delta) expression correctly prevents negative inventory values under all adjustment scenarios. Usability testing confirmed that all five participants completed the full administrator workflow — login, product creation, stock adjustment, and audit log review — without any technical assistance, validating the accessibility objective of the system.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

VIII. CONCLUSION

This paper presented Uppi's Restaurant Store Management System, a full-stack web application developed using Next.js 14, Supabase PostgreSQL, TypeScript, and Tailwind CSS, designed to eliminate the inventory management deficiencies faced by small to mid-sized restaurant operators in India. The system delivers real-time stock visibility through colour-coded status indicators, complete movement traceability through a structured audit log, and aggregate financial visibility through a live inventory value KPI.

All 14 functional test cases passed successfully. The system meets all defined performance benchmarks, including sub-800ms dashboard load times and under-200ms stock adjustment responses. The entire deployment operates at zero monthly infrastructure cost using Vercel and Supabase's free tiers. OTP-based authentication via Supabase Auth provides production-grade security without custom credential management.

The system demonstrates that a production-ready, feature-complete restaurant inventory platform can be built using modern open-source technologies without financial investment in infrastructure — directly addressing the cost, complexity, and technical expertise barriers that have historically prevented independent restaurant operators from adopting digital inventory systems.

IX. FUTURE SCOPE

- Multi-User Role Management: Extend the system to support kitchen staff (movement logging only) and managers (read-only reporting) alongside the administrator role, implemented through Supabase RLS policies scoped by user role.
- Automated Low-Stock Email Alerts: A Supabase Edge Function scheduled to run daily, querying products below minimum threshold and dispatching email alerts via SendGrid or Resend.
- POS System Integration: API integration with popular Indian POS systems (Petpooja, Posist) to automatically deduct ingredient quantities from inventory when menu items are sold, creating a real-time consumption model.
- Automated Purchase Order Generation: Generating PDF purchase orders pre-populated with supplier contact details and suggested quantities when stock falls below minimum thresholds.
- Progressive Web App (PWA): Converting the Next.js frontend to a PWA for home screen installation and mobile-optimised use on smartphones without requiring a native app.
- Barcode / QR Scanning: Adding camera-based barcode scanning for rapid stock-in recording by scanning product barcodes at the time of delivery, eliminating manual SKU entry.

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 97-111

- Advanced Analytics Dashboard: Recharts-powered charts for stock consumption trends, waste percentage by category, inventory value history, and supplier performance metrics over time.
- Multi-Location Support: Extending the schema to support multiple restaurant branches with location-specific inventory and cross-location stock transfer logging.

REFERENCES

[1] National Restaurant Association, "State of the Restaurant Industry Report," National Restaurant Association, Washington, D.C., 2023.

[2] L. R. Dopson and D. K. Hayes, Food and Beverage Cost Control, 5th ed. Hoboken, NJ: John Wiley & Sons, 2010.

[3] R. Engström and A. Carlsson-Kanyama, "Food losses in food service institutions — Examples from Sweden," Food Policy, vol. 29, no. 3, pp. 203–213, Jun. 2004.

[4] R. Srinivasan, M. Krishnamurthy, and S. Patel, "Web-based Inventory Management Systems for Small Businesses: A Survey," International Journal of Information Management, vol. 45, pp. 102–118, 2019.

[5] A. Kumar and P. Sharma, "Adoption Barriers for Digital Inventory Systems in Indian Restaurant Operations," Journal of Hospitality Technology, vol. 12, no. 2, pp. 45–62, 2021.

[6] Vercel, Inc., "Next.js 14 Documentation — App Router," 2024.

[7] Supabase, Inc., "Supabase Documentation — Database, Auth, and Storage," 2024.

[8] Microsoft, "TypeScript 5 Handbook," 2024.

[9] Tailwind Labs, "Tailwind CSS v3 Documentation," 2024.

[10] The PostgreSQL Global Development Group, "PostgreSQL 15 Documentation," 2024.

[11] OWASP Foundation, "OWASP Top Ten Web Application Security Risks," 2021.

[12] Recharts, "Recharts — Redefined chart library built with React and D3," 2024.