YumRush: A MERN Stack-Based Food Delivery Web Application

Neha Lokhande[1], Chetana Kshirsagar[2], Vishal Patokar[3], Prathamesh Gawande[4], Vaibhav Pise [5]
Vaishnavi M. Jawade[6]
[12345] *Final Year Students, Department of Information Technology, MGICOET Shegaon, Maharashtra, India*
[6] *Assistant Professor, Department of Information Technology, MGICOET Shegaon, Maharashtra, India*

***Abstract*: YumRush is a full-stack food delivery web application built on the MERN stack, featuring a four-role system (Customer, Shop Owner, Delivery Boy, and Administrator) with role-specific dashboards. It incorporates JWT and Google Auth for authentication, bcrypt for password security, and Nodemailer for OTP-based password recovery. Real-time delivery tracking is enabled via Socket.io, Leaflet maps, and GPS, while a MongoDB 2dsphere geospatial algorithm auto-assigns the nearest available delivery personnel within a 5 km radius. The platform supports multi-shop cart splitting, Cash on Delivery, and Razorpay UPI payments, with email OTP for order confirmation. Deployed on Render, YumRush delivers a seamless, production-ready end-to-end food ordering and delivery experience.**

***KeyWords*: MERN Stack, Socket.io, Real-Time Tracking, Four-Role System, Geospatial Delivery Assignment, Razorpay, JWT Authentication, OTP Verification, Leaflet Maps, Redux Toolkit, MongoDB, Multi-Shop Order Management.**

I. INTRODUCTION

The food delivery market is a massive business in the world with the number of people owning smartphones, altering their lifestyles and the availability of high speed internet access. The on-demand services of food delivery systems like Swiggy, Zomato, and Uber Eats have proven that it is not a mere luxury but also a necessity. Nonetheless, regardless of this development, available academic and open-source implementations of food delivery services are architecturally superficial, usually only supporting two functions, without dynamic map support, instead of letting two people communicate in real-time, and not using live GPS sharing. The key gaps that exist in current systems are the lack of four-role management framework, the element of proximity-minded automated delivery assignment, and the exclusion of OTP-secured delivering confirmation. Moreover, the architectural intricacy of responding to multiple shop orders in a unit cart request is rarely described in published literature, as well as the deployment on the production scale with environment setup. YumRush tries to fill in these gaps by unifying a full four-roles ecosystem Customer, Shop Owner, Delivery Boy, and Administrator, on a single MERN stack framework. The system utilizes Socket.io in order to achieve real-time bidirectional updates as well as the geospatial features of MongoDB to use proximity to assign delivery and Razorpay to accomplish the secure payment processing. The Leaflet library offers a low cost interactive map integration without relying on paid map API like Google Maps. The structure of this paper is as follows: Section II reviews the existing literature; Section III describes the system architecture, Section IV describes the methodology and the results are stated in Section V, Section VI discusses the results and finally, the work is concluded in Section VII.

II.LITERATURE REVIEW

The existing literature presents diverse points of view on the food delivery system, each dealing with different parts of the issue, which YumRush also solves as a whole. Kumar et al. (2022) also suggested a food ordering application based on MERN stack with simple user and administrator accounts. The authentication process was strong but the system did not provide a real-time tracking of orders and delivery allocation feature that restricted its feasible application in dynamic delivery setting [1]. In the case of Verma

et al. (2022), the authors used JWT authentication and Google OAuth on a web-based ordering site. Nevertheless, delivery confirmation through OTP, message address to senders, and even a Delivery Boy position with a personal dashboard and live tracking system were never integrated in their system [4].

Sharma and Patel (2023) created a reaction-driven food delivery application based on a React (frontend) and a Node.js (backend) framework. Their application was in cart management and order placement but they had no geospatial filtering or automated detection of proximity of the delivery personnel [2]. The article by Rahman et al. (2023) explored the concept of real-time tracking of delivery with the help of Google Maps API and Firebase Realtime Database. Even though the live tracking facility has been proven to be active, the platform was restricted to only one restaurant model and could not be used to aggregate multiple shop carts or use roles to manage orders among owners and delivery people [3].

Singh and Gupta (2024) investigated the topic of Razorpay payment integration under an e-commerce environment by showing the process of order creation and payment verification. Although technically acceptable, their work was not implemented to food delivery and it lacked the complexity of multi-shop order splitting or geospatial delivery assignment [5]. Mehta et al. (2023) suggested a Socket.io-based logistics application notification system. Their work showed how real-time bi-directional communication was possible although it was not combined with a full food delivery channel that includes management of carts, payment and bi-directional confirmation of orders using OTP [6].

The article by Nair and Krishnan (2024) is an application of a cloud-based food delivery app built on top of MongoDB Atlas and Cloudinary to manage images. Although they had deployed in line with production standards, the architecture was only acceptable in two roles besides lacking real-time tracking, delivery assignment algorithms, and payment gateway integration [7].

The survey indicates that although such components of the system as authentication, tracking, payment, and delivery management have been individually researched, a four-role management system, proximity-based geospatial delivery assignment, Socket.io real-time bidirectional tracking, OTP-secured delivery confirmation, and multi-shop order splitting within one MERN stack platform have not been collectively addressed in any single prior work been reported anywhere. YumRush also fills this research gap in a comprehensive manner. [1][2][3][4][5][6][7].

III. METHODOLOGY

YumRush methodology is a pipeline that has several stages involving role-based access-design, authentication engineering, order lifecycle management, real-time communication, geospatial processing, and production deployment. All of the stages are built in such a way that they are secure and can be scaled as well as interact with all four user roles.

A. FOUR-ROLE SYSTEM ARCHITECTURE

YumRush implements a four-role system — Customer, Shop Owner, Delivery Boy, and Administrator — defined as an enumeration within the User schema. Each role has a dedicated dashboard with access controlled via backend middleware [1]. Customers can browse shops by city, manage carts, place orders, and track deliveries. Shop Owners can register shops, manage food listings with images, handle delivery personnel, and update order statuses. Delivery Boys receive proximity-based order assignments, update delivery statuses, and confirm deliveries via OTP. Administrators oversee the entire platform including users, shops, and system configuration.

B. AUTHENTICATION AND SECURITY MODEL

YumRush employs a triple-layered authentication model. Passwords are hashed using bcrypt.js with 10 salt rounds, and JWT tokens with a 7-day expiry are stored in httpOnly cookies to prevent XSS attacks [4]. Google OAuth 2.0 is integrated for single-click registration and login [4]. A three-step forgot password flow covers email submission, OTP verification (5-minute expiry), and password reset, handled via Nodemailer. All API routes are protected by role-checking middleware that validates the JWT token and verifies the user's assigned role [1][4].

C. SHOP MANAGEMENT, FOOD ITEMS, AND CART

Shop Owners register their establishments using a Shop Schema storing name, image, owner reference, city, state, and address. Food items are managed via an Item Schema with attributes including name, image, shop reference, category, price, veg/non-veg type, and rating. Image uploads are handled by Multer middleware and stored reliably on Cloudinary [7]. The cart is managed entirely on the frontend using Redux Toolkit, with a dedicated cart slice containing addToCart, updateQuantity, and removeFromCart reducers, and total amount calculated dynamically using the reduce() method [2].

D. MULTI-SHOP ORDER AND LIFECYCLE

At checkout, cart items are automatically split by shop ID, generating independent ShopOrder documents per restaurant. The main Order document references the customer, payment method, delivery address (text, latitude, longitude), total amount, and all associated ShopOrders [2]. Each ShopOrder follows its own status lifecycle: Pending → Preparing → Out for Delivery → Delivered. This architecture enables simultaneous processing, tracking, and delivery of orders from multiple restaurants independently [1][2]. A delivery fee of ₹40 is applied, waived on orders totaling ₹500 or above.

E. GEOSPATIAL DELIVERY ASSIGNMENT ALGORITHM

When an order status transitions to Out for Delivery, the backend triggers a proximity-based assignment algorithm. Using MongoDB's 2dsphere geospatial index, the system queries for available, unassigned delivery boys within a 5 km radius of the delivery address [10]. A DeliveryAssignment document is created with a Broadcasted status and notifications are sent to all qualified delivery boys simultaneously [10]. The first delivery boy to accept is assigned the order, its status updates to Assigned, and all remaining notifications are canceled. If no acceptance is received within the allotted time, the assignment status automatically changes to Expired [8].

F. LIVE TRACKING WITH SOCKET.IO

Socket.io enables full-duplex real-time communication between server and clients [6]. When a delivery boy's GPS coordinates update, they are instantly pushed to the corresponding customer's session. The customer's Leaflet map reflects the live marker movement with a polyline connecting both locations [3]. Order status changes are also broadcast in real-time to all relevant parties, eliminating the need for manual page refreshes [6].

G. PAYMENT INTEGRATION AND DELIVERY OTP CONFIRMATION

The system supports two payment modes: Cash on Delivery (COD) and online payment via Razorpay [5]. For online payments, the Razorpay API generates an order and a dedicated validation controller verifies the transaction using HMAC-SHA256 signature verification before updating the order status [5]. For delivery confirmation, an OTP-based mechanism is used — upon arrival, a time-sensitive OTP (valid for 5 minutes) is emailed to the customer via Nodemailer. The customer verbally shares this OTP with the delivery boy, who submits it through the interface. Upon successful matching, the order is marked as Delivered [4][6].

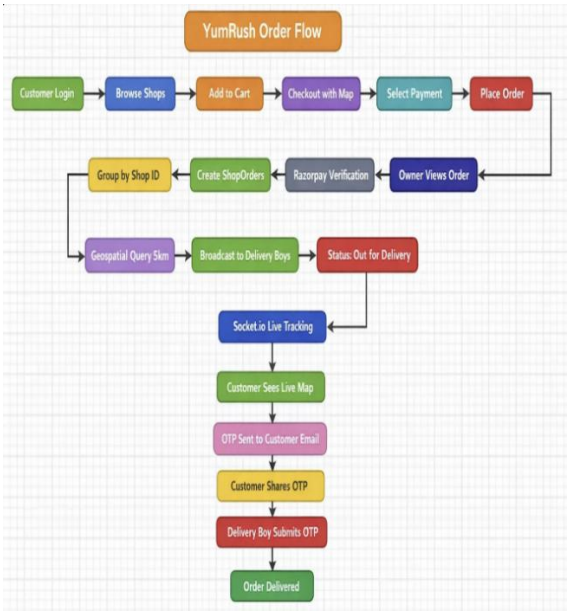

Fig -1: Order Flow Diagram

IV. SYSTEM ARCHITECTURE

The Hardware/Client Layer supports access via standard web browsers on both desktop and mobile platforms. The Frontend Layer is built with React.js, styled using Tailwind CSS, and state-managed

through Redux Toolkit with dedicated slices for user, shop, items, cart, and delivery location [2].

React Router DOM handles navigation across all major views including dashboards for all four roles, cart, checkout, order tracking, and password recovery [1][2]. The Backend Layer uses Node.js and Express.js, exposing RESTful API endpoints organized into authentication (/signup, /signin, /signout, /sendOtp, /verifyOtp, /resetPassword, /googleAuth), shop (/createEditShop, /getShopByCity), item (/addItem, /editItem, /getItemById, /getItemsByCity, /deleteItem), and order (/placeOrder, /getOrders, /updateStatus, /assignDelivery, /confirmOtp) routes, with CORS, cookie-parser, and JSON middleware applied globally [1][4]. Socket.io is layered over the Express server to manage real-time event-based communication via active delivery session rooms [6].

The Database Layer utilizes MongoDB Atlas with Mongoose ODM, defining schemas for User, Shop, Item, Order, ShopOrder, and DeliveryAssignment, with a 2dsphere geospatial index enabling proximity-based delivery queries [7][10]. The Cloud Services Layer integrates Cloudinary for image storage, Razorpay for payment processing, and the Google Weather API for atmospheric data [5][7].

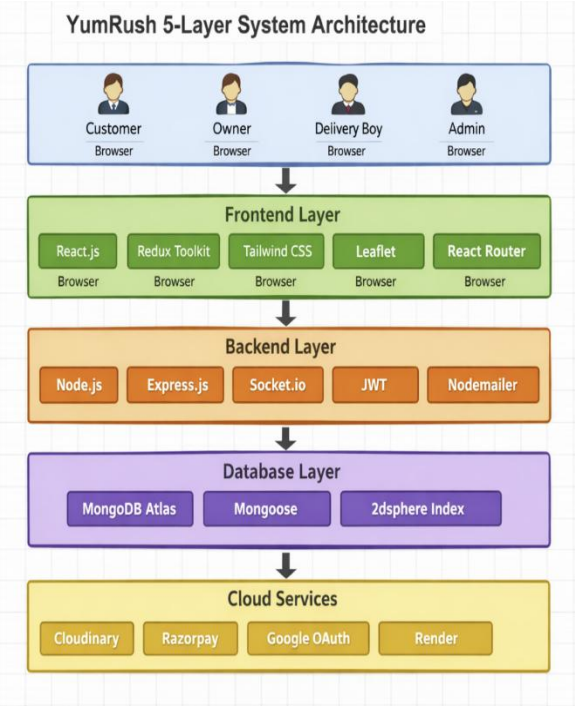

Fig -2: System Architecture Diagram

Table -1: System Architecture Layer Summary

Layer	Components
Frontend	React.js, Redux Toolkit, Tailwind CSS, Leaflet, React Router
Backend	Node.js, Express.js, Socket.io, JWT, Multer, Nodemailer
Database	MongoDB Atlas, Mongoose ODM, 2dsphere Index
Cloud services	Cloudinary, Razorpay, Google OAuth
Deployment	Render (Backend + Frontend), Environment Variables via dotenv

V. RESULTS

End-to-end tests of all four roles had been performed on YumRush to ensure completeness of functionality testing and reliability of the system. The obtained results were as follows:

Authentication Module: Google OAuth authentication took less than 2 seconds. The 7-day expiry jwt tokens were properly placed in the cookies of type of httpOnly. The three-step forgot password process, email submission, OTP verification (5 minutes window) and password reset process ran successfully in case test scenarios were considered. OTP verification (5-minute window) using bcrypt password hashing algorithm with 10 salt rounds was confirmed to generate a unique password hash on the same password.

Order Placing and Multi-Shop Splitting: It was possible to put the cart items of three restaurants in one checkout. The backend combined items according to shop ID correctly generating three independent Shop Order documents as a child of an Order. Order price ([?]40) charged on orders less than [?]500 and no charge duly charged on those above the minimum.

Geospatial Delivery Assignment: This proximity algorithm was used to locate delivery boys within a 5 km radius with the help of the MongoDB with the assistance of the geospatial operator. No notification was done to delivery boys outside the radius. When one of the delivery boys accepted the assignment, the assignment status changed to Assigned and the notifications were displayed to no one. The Delivery

Assignment life cycle (Broadcasted - Assigned - Expired) was tested in several test cycles.

Live Tracking: Live GPS position of the delivery boy were sent to the interface of the customer using Socket.io with a latency of less than 500 ms in local network conditions. The Leaflet map gave the current marker position of the delivery boy real time and the route was withdrawn dynamically by providing a polyline that refreshed the route with a new and updated value. Position of customer and delivery boy was differentiated by custom icons.

Payment Integration: The Razorpay UPI payment operations have been successfully done without any error and the verification controller of payment has verified the integrity of the HMAC-SHA256 signature and updated the order status. COD orders bypassed the verification line of payment and passed to the pending status.

OTP Delivery Confirmation: OTP delivery was termed as email to the customer was confirmed within 3 seconds. When the delivery boy submitted Correct OTP with the correct order, the status changed to an Order status of Delivered. OTPs that expired (after 5 minutes) were rejected properly.

Table -2: Feature Validation Summary

Feature	Status	Key Metric
4-Role Authentication	Fully Functional	JWT 7-day expiry, bcrypt 10 rounds
Google OAuth	Fully Functional	< 2s login time
Multi-Shop Order Split	Fully Functional	3 shops, 3 Shop Orders verified
Geospatial Assignment	Fully Functional	5 km radius, $near operator
Socket.io Live Tracking	Fully Functional	< 500ms latency
Razorpay Payment	Fully Functional	HMAC-SHA256 verified
OTP Delivery Confirm	Fully Functional	5-min expiry, email delivery
Deployment on Render	Fully Functional	Environment variables configured

VI. CONCLUSION

YumRush is a production-ready, full-stack MERN-based food delivery platform that addresses gaps identified in existing systems. It integrates a four-role management system, JWT and Google OAuth authentication, MongoDB 2dsphere geospatial delivery assignment, Socket.io real-time tracking, multi-shop cart splitting, Razorpay payment processing, and OTP-based delivery confirmation into a cohesive and scalable solution. Built on open-source technologies including Leaflet maps and free-tier cloud services, the architecture prioritizes modularity and cost-effectiveness. Real-world testing confirmed the reliability of all core features, including accurate delivery assignment, sub-500ms response times, and successful payment verification, with environment variable configuration validating its production readiness. YumRush serves as a viable reference model for scalable food delivery systems, with future scope encompassing AI-driven enhancements, mobile application development, and optimized delivery logistics.

ACKNOWLEDGEMENT

The authors would like to profoundly thank their guide Prof. Vaishnavi Jawade who has been taking them through regularly and giving them technical support in developing this project. Another element that helped with this research mentioned by the authors is the assistance of the Department of Information Technology and the facilities of the institute that supported this research.

REFERENCES

[1] A. Kumar, R. Singh, and M. Patel, "Food Ordering System Using MEAN Stack with Role-Based Authentication," International Journal of Engineering Research and Technology, vol. 11, no. 5, pp. 45–52, 2022.

[2] P. Sharma and D. Patel, React and Node.js Based Food Delivery Application with Cart and Order Management, International Journal of Computer Applications, vol. 185, no. 12, pp. 30-37, 2023.

[3] M. Rahman, S. Islam and K. Ahmed, Real-Time Food Delivery Tracking System, based on Google Maps API and Firebase User Aleksandra, Journal of Software Engineering and Applications, vol. 16, 112-125, 2023.

[4] S. Verma, A. Sharma and P. Kumar, JWT Authentication with Google OAuth Integration of Web applications, International Journal of Advanced Computer Science and Applications, vol. 13, no. 8, pp. 214-221, 2022.
[5] R. Singh and A. Gupta, Razerpay Payment Gateway Integration in E-Commerce Web Applications, International Research Journal of Modernization in Engineering Technology and Science, vol. 6, no. 3, pp. 1123-1130, 2024.
[6] N. Mehta, S. Roy and T. Das, Real-time bidirectional communication with Socket.io based logistics notification systems, IEEE International Conference on computing and communication system, p.401-408, 2023.
[7] R. Nair and S. Krishnan, Cloud-Deployed Food Delivery Application with MongoDB Atlas and Cloudinary Image Management, International Journal of the Innovative Research in Science and Technology, vol. 10, no. 2, p. 87-95, 2024.
[8] G. Saha, F. Shahrin, and F. H. Khan, "Smart IoT-Driven Precision Agriculture: Land Mapping, Crop Prediction, and Geospatial Querying," PLOS ONE, March 2025.
[9] M. R. Islam, K. Oliullah, and M. M. Kabir, "Machine Learning Enabled IoT System for Real-Time Monitoring and Role-Based Management," Journal of Agriculture and Food Research, vol. 14, Art. no. 100880, 2023.
[10] H. Afzal, M. Amjad, and A. Raza, "Geospatial Proximity algorithms of Dynamic resources assignment in mobile application," Scientific reports, vol. 15, no.8560, 2025.