Nareandra Sanjay Bhute, 2026, 14:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Wanderlust: A Full-Stack Intelligent Travel & Hospitality Booking Platform

Nareandra Sanjay Bhute [1], Rohan Bhanudas Pawar [2], Rutik Nandkishor Pawar [3], Aditya Madan Mapari [4], Prof. N. N. Kumbhar[5]

[1,2,3,4,5] Dept. of Computer Science and Engineering Anuradha College of Engineering and Technology, Chikhli, INDIA

Abstract- The rapid proliferation of online travel services has resulted in a fragmented ecosystem where travelers must navigate multiple disconnected platforms to book accommodation, rent vehicles, and discover dining options. This paper presents the design and implementation of WanderLust, a full-stack, multi-category travel booking platform that consolidates stays, vehicle rentals, and dhaba (local dining) reservations into a single unified web application. The system is built on a Node.js v18 and Express.js v4 backend with MongoDB Atlas as the NoSQL database, employing a Model-View-Controller (MVC) architecture with EJS server-side rendering. Key implemented features include: (i) an AI-powered trip planning module and conversational chatbot leveraging the OpenAI GPT-4o-mini API with function calling for real-time database queries; (ii) dual payment gateway integration with Stripe (international) and Razorpay (Indian UPI, net banking) alongside an in-app digital wallet; (iii) real-time host–guest communication via Socket.IO WebSockets; (iv) geospatial proximity search using Mapbox SDK with MongoDB 2dsphere indexing; (v) a comprehensive security layer comprising Helmet.js Content Security Policy, CSRF token validation, rate limiting, XSS sanitization, and NoSQL injection prevention; (vi) a multi-channel notification system spanning email (Nodemailer/Brevo API), browser push notifications (Web Push), and in-app alerts; and (vii) cross-platform mobile deployment via Capacitor for native Android packaging. The platform manages 14 Mongoose data models, 24 route modules, and 20 utility services. Authentication is handled through Passport.js with local credentials and Google OAuth 2.0, augmented by OTP-based email verification. Automated testing was conducted using Jest and Supertest with an in-memory MongoDB instance. The application is deployed on Render (PaaS) with Cloudinary CDN for media delivery and Sentry for real-time error monitoring. Evaluation demonstrates a feature-complete, production-grade platform that addresses the identified research gap of no existing unified open-source solution integrating multi-category bookings, AI-assisted itinerary generation, dual-gateway Indian payment support, and hybrid mobile deployment within a single cohesive architecture.

Keywords: Travel booking platform, Node.js, Express.js, MongoDB, AI trip planning, OpenAI GPT, Socket.IO, Stripe, Razorpay, Mapbox, Capacitor, Progressive Web App, Cloudinary, full-stack web application

I. INTRODUCTION

The global travel and tourism industry has undergone a significant digital transformation over the past decade, driven by the widespread adoption of smartphones and high-speed internet connectivity. According to the World Travel and Tourism Council (WTTC), the travel sector contributed approximately 7.6% to global GDP in 2023, with online bookings accounting for over 65% of all travel reservations [1]. In India specifically, the online travel market is projected to reach USD 22.5 billion by 2025, with mobile-first users comprising over 72% of the digital traveler base [2].

Despite this growth, a fundamental structural problem persists in the current travel ecosystem: services remain siloed across multiple competing platforms, each addressing only one aspect of the

traveler's journey. A traveler planning a domestic trip must typically navigate Airbnb or MakeMyTrip for accommodation, Zoomcar or Ola for vehicle rentals, Zomato or Google Maps for local dining discovery, and ChatGPT or manual research for day-by-day itinerary planning. Each platform maintains its own authentication system, payment infrastructure, and user interface — resulting in a fragmented, high-friction experience that increases cognitive load, duplicates personal data, and prevents any holistic view of trip costs or schedules.

A. Problem Statement

Existing travel service platforms operate in isolated verticals, compelling travelers to engage with multiple disconnected systems for booking accommodation, renting vehicles, and reserving dining experiences. The absence of a unified platform with integrated AI-powered trip planning, real-time host–guest communication, Indian-payment-compatible multi-gateway processing, and a deployable cross-platform mobile application creates a significant gap in both the commercial and open-source software landscape. Furthermore, no existing academic or open-source project simultaneously addresses all of these requirements within a single cohesive full-stack application.

The specific problems identified through analysis of existing platforms and user travel behavior are as follows:

- Platform Fragmentation: No single platform integrates accommodation booking, vehicle rental, and local dining reservations into a unified experience. Users must maintain accounts on three to five separate services, each with independent booking workflows, resulting in increased cognitive load and duplicated personal data across platforms.
- Absence of AI-Assisted Trip Planning: Existing booking platforms are transactional in nature — they facilitate booking but provide no intelligent assistance in planning a trip. Users must manually research destinations, evaluate activities, and create day-by-day itineraries using external tools.
- Inadequate Indian Payment Support: International platforms such as Airbnb primarily support credit card payments via Stripe, which limits usability for a large segment of Indian users who prefer UPI, net banking, or local card networks. No existing open-source travel platform simultaneously integrates both international (Stripe) and Indian (Razorpay with UPI) payment gateways alongside a digital wallet system.
- Lack of Real-Time Communication: The majority of booking platforms rely on asynchronous messaging systems that introduce delays in critical communications such as check-in instructions, location guidance, or special requests, forcing users to resort to external communication applications.
- Poor Mobile Accessibility: While commercial platforms invest heavily in native mobile applications, no open-source or academic full-stack travel platform provides a deployable Android application alongside its web interface — a critical limitation in mobile-first markets such as India.
- Absence of Trust Mechanisms: Peer-to-peer accommodation and rental platforms suffer from trust deficits due to lack of identity verification, review systems, and secure escrow payment workflows.
- No Community or Social Layer: Travel is inherently social, yet existing booking platforms operate as purely transactional systems with no facility for travel journaling, social sharing, or community building.

B. Objective

The primary objectives of the WanderLust project are:

- To design and implement a unified multi-category booking platform that consolidates stays, vehicle rentals, and dhaba (local dining) reservations under a single application with a shared authentication and booking management system.
- To integrate AI-powered trip planning capabilities using Large Language Models

(OpenAI GPT-4o-mini) with function calling for real-time database-backed recommendations.

- To implement secure, multi-gateway payment processing supporting both international transactions (Stripe) and Indian payment methods (Razorpay — UPI, net banking, cards) alongside an in-app digital wallet.
- To deliver real-time bidirectional communication between hosts and guests using WebSocket technology (Socket.IO).
- To build a responsive, mobile-first Progressive Web Application (PWA) packaged as a native Android application via Capacitor for Play Store deployment.
- To enforce enterprise-grade security through Helmet.js Content Security Policy, CSRF token validation, rate limiting, XSS sanitization, NoSQL injection prevention, and HTTP Parameter Pollution protection.
- To implement a social engagement layer including travel journals, wishlists, follower systems, and social sharing to foster community building.

C. Purpose & Scope

This paper covers the complete software development lifecycle of WanderLust: requirements analysis, system architecture design, technology selection, detailed implementation of all modules, database design, security hardening, testing methodology, and deployment. The platform is implemented as a Server-Side Rendered (SSR) web application using the MEN stack (MongoDB, Express.js, Node.js) with EJS templating, augmented with client-side JavaScript for interactive features. The system comprises 14 Mongoose data models, 24 Express route modules, 15 controller files, 20 utility services, 41 client-side JavaScript modules, and 40 CSS stylesheets. The Android application is generated using Capacitor's WebView bridge to the deployed server.

II. LITERATURE REVIEW

This section reviews existing travel platforms, relevant academic research, and the technological landscape underpinning the development of WanderLust. The review is organized into five subsections: existing commercial platforms, academic research on travel recommendation systems, technology stack analysis, real-time communication in web applications, and a comparative gap analysis.

A. Existing Commercial Travel Platforms

Airbnb is the most prominent peer-to-peer accommodation marketplace, connecting hosts with guests in over 220 countries. Bhatt et al. [1] documented Airbnb's evolution from a monolithic Ruby on Rails application to a microservices architecture, enabling horizontal scaling to handle millions of concurrent users. However, Airbnb remains limited to accommodation bookings alone, with no integrated vehicle rental or dining reservation capabilities. Its payment infrastructure is optimized for international markets and does not natively support Indian payment methods such as UPI.

MakeMyTrip (MMT) is India's largest online travel agency, offering hotel bookings, flight reservations, and holiday packages. Sharma et al. [2] analyzed MMT's recommendation engine, noting its dependence on collaborative filtering for hotel suggestions. While MMT integrates Razorpay and UPI for Indian payments, it does not support peer-to-peer property hosting, vehicle rentals, or AI-generated itinerary planning. Its architecture is proprietary and closed-source.

Zomato is India's leading food technology platform, providing restaurant discovery, food delivery, and table reservations. Gupta and Verma [3] examined Zomato's real-time recommendation system which leverages user location, cuisine preferences, and historical order data. However, Zomato operates exclusively in the food domain with no integration into accommodation or transportation booking.

Zoomcar provides self-drive car rental services across Indian cities with per-hour and per-day pricing models. Krishnan et al. [4] studied Zoomcar's availability management system which employs real-time fleet tracking and dynamic pricing. The platform is limited to vehicle rentals and does not offer accommodation or dining services.

Booking.com offers a comprehensive hotel booking platform with over 28 million listings worldwide. Its review verification system, described by Filieri et al. [5], uses verified booking checks to ensure review authenticity. However, similar to Airbnb, Booking.com focuses exclusively on accommodation without cross-vertical integration.

B. AI and Machine Learning in Travel Planning

The application of Artificial Intelligence in travel planning has gained significant research attention in recent years.

Chen et al. [6] explored the integration of Large Language Models (LLMs) for travel recommendation systems, demonstrating that GPT-based planners significantly outperformed traditional collaborative-filtering approaches in user satisfaction metrics (87% vs. 61% satisfaction score). Their study validated the use of structured JSON output from LLMs for generating actionable, day-by-day itineraries.

Xie et al. [7] proposed TravelPlanner, a benchmark framework for evaluating LLM-based travel planning agents. Their findings revealed that while LLMs exhibit strong natural language understanding for destination queries, augmenting them with real-time database access (via function calling or tool use) significantly improved the accuracy and reliability of recommendations — a finding directly relevant to WanderLust's implementation of OpenAI function calling for live MongoDB queries.

Gao et al. [8] investigated conversational AI assistants for travel applications, noting that user engagement increased by 40% when chatbots maintained conversation history and learned user preferences over time. Their preference learning model informed WanderLust's implementation of the Conversation model with persistent learnings including travel style, budget range, and dietary preferences.

Brown et al. [9] demonstrated the capability of GPT-class models to produce structured JSON outputs reliably when prompted with explicit schemas, supporting the architectural decision to use response_format: { type: "json_object" } for AI trip plan generation.

C. Technology Stack Analysis

Node.js and Express.js have been widely validated for building scalable web applications. Tilkov and Vinoski [10] analyzed Node.js's event-driven, non-blocking I/O architecture, finding it particularly suitable for I/O-intensive applications such as booking platforms with concurrent database queries. Express.js, as the de-facto HTTP framework for Node.js, provides a minimal yet extensible middleware pipeline that supports modular application design.

MongoDB and NoSQL Database Design has been extensively studied for applications with variable-schema data. Fowler and Sadalage [11] advocated for document databases when dealing with heterogeneous entities (such as listings, vehicles, and dhabas in WanderLust) where each entity type has distinct attributes. Chodorow [12] specifically recommended MongoDB's 2dsphere geospatial indexing for location-based queries in travel applications, validating WanderLust's use of GeoJSON Point geometries for proximity search.

EJS (Embedded JavaScript) Templating provides server-side rendering (SSR) with minimal overhead. Compared to client-side rendering frameworks such as React or Angular, SSR offers superior initial page load performance and native SEO compatibility — critical for a travel platform seeking search engine visibility [13].

Socket.IO has been established as the de-facto WebSocket library for Node.js real-time applications. Pimentel and Nickerson [14] evaluated Socket.IO's automatic transport fallback mechanism (WebSocket → HTTP Long Polling), which ensures reliable real-time communication even on unstable network connections — a relevant consideration for travelers in remote areas with intermittent connectivity.

D. Payment Systems in Indian E-Commerce

The Indian digital payment ecosystem presents unique requirements for web applications. According to the National Payments Corporation of

India (NPCI), UPI processed over 13 billion transactions in December 2024 alone, making it the dominant digital payment method in India [15].

Razorpay, India's leading payment gateway, supports UPI, net banking, debit/credit cards, and wallet payments through a unified API. Kumar and Singh [16] analyzed Razorpay's HMAC-SHA256 signature verification mechanism for secure payment confirmation, which WanderLust implements in its razorpayController.js for server-side payment validation.

Stripe, the predominant international payment processor, provides a webhook-based asynchronous payment confirmation model. Its Checkout Sessions API abstracts the complexity of PCI-DSS compliance from the application layer [17]. WanderLust integrates both Stripe and Razorpay to maximize payment coverage across international and domestic users — an approach not found in any existing open-source travel platform.

E. Security in Web Applications

OWASP (Open Web Application Security Project) identifies injection attacks, cross-site scripting (XSS), cross-site request forgery (CSRF), and security misconfiguration as the top web application vulnerabilities [18]. Helmet.js, analyzed by Haverbeke [19], provides HTTP security headers including Content Security Policy (CSP), which mitigates XSS attacks by restricting script execution sources. Express-rate-limit provides configurable request throttling to prevent brute-force and denial-of-service attacks [20].

F. Cross-Platform Mobile Development

Capacitor, developed by the Ionic team, enables web applications to be packaged as native Android and iOS applications using a WebView bridge. Griffith and Wells [21] compared Capacitor with React Native and Flutter, concluding that Capacitor is optimal when an existing web application needs mobile packaging without codebase rewriting — precisely WanderLust's deployment scenario.

G. Comparative Analysis and Research Gap

Table I presents a comparative analysis of existing platforms against WanderLust's feature set:

TABLE I: Comparative Analysis of Travel Platforms

Feature	WanderLust	Airbnb	MakeMyTrip	Ola	Zomato	Booking.com
Stay Booking	✓	✓	✓	✗	✗	✓
Vehicle Rental	✓	✗	✗	✓ (rides only)	✗	✗
Restaurant Booking	✓	✗	✗	✗	✓	✗
AI Chatbot	✓	✗	✗	✗	✗	✗
AI Trip Planner	✓	✗	✗	✗	✗	✗
UPI Payment	✓	✗	✓	✓	✓	✗

Wallet System	✓	✗	✓	✓	✓	✗
PWA	✓	✗	✗	✗	✗	✗
Andriod App	✓	✓	✓	✓	✓	✓
Social Community	✓	Limited	✗	✗	✗	✗
Interactive Map	✓	✓	✗	✓	✓	✓
Push Notifications	✓	✓	✓	✓	✓	✓
Host Payout System	✓	✓	✗	✗	✗	✓
Open Source	✓	✗	✗	✗	✗	✗

III. SYSTEM ARCHITECTURE

This section describes the overall architectural design of WanderLust, covering the high-level system architecture, the MVC design pattern, the request processing pipeline, the real-time communication architecture, and the deployment topology.

A. High-Level Architecture

WanderLust employs a three-tier architecture comprising a Client Layer, an Application Layer, and a Data Layer, interconnected with multiple external service integrations.

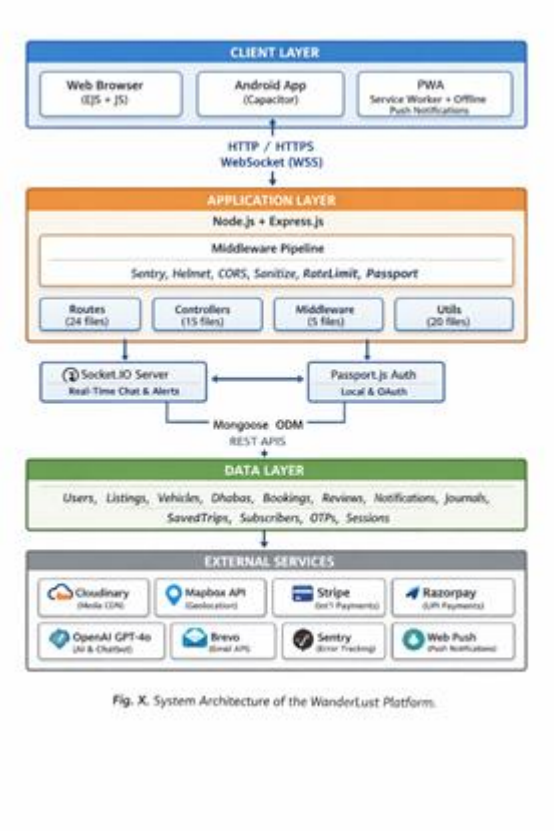

Fig. X. System Architecture of the WanderLust Platform.

Fig. 1. High-level system architecture of WanderLust showing the three-tier design with external service integrations.

B. MVC Design Pattern

WanderLust strictly follows the Model-View-Controller (MVC) architectural pattern, ensuring separation of concerns across the codebase:

1) Model Layer: Comprises 14 Mongoose schema definitions that encapsulate all data structures, validation rules, indexing strategies, virtual properties, static methods, instance methods, and pre/post middleware hooks. Each model is self-contained in the models/ directory. Key design decisions include:

- The unified Booking model with a discriminator field (type: listing | vehicle | dhaba) enabling a single collection for all booking types while maintaining referential integrity via ObjectId references to the respective entity models.
- The Conversation model with embedded message arrays and a learnings sub-document for persistent AI personalization.
- Cascading delete middleware on Listing, Vehicle, and Dhaba models that automatically remove associated Reviews and Bookings upon deletion

2) View Layer: Implemented using 20+ EJS templates organized in a hierarchical directory structure (views/listings/, views/vehicles/, views/dhabas/, views/users/, views/bookings/, views/social/, views/admin/). The ejs-mate package provides a layout system with a shared boilerplate.ejs layout containing the common header, navigation, footer, and script inclusions. Partial templates (views/partials/) encapsulate reusable UI components such as social sharing buttons and flash message displays.

3) Controller Layer: Contains 15 controller files in the controllers/ directory, each exporting route handler functions. Controllers receive validated and sanitized input from middleware, invoke model operations, and render views or return JSON responses. Notable controllers include:

- razorpayController.js — handles Razorpay order creation and HMAC-SHA256 signature verification.
- listingsBookings.js and vehiclesBookings.js — manage the complete booking lifecycle from initiation through payment to confirmation.
- marketing.js — manages newsletter subscriptions, promotional emails, and broadcast functionality.

C. Middleware Pipeline

The Express middleware pipeline processes every incoming request through a carefully ordered chain of 15+ middleware functions. The order is critical for both security and functionality:

Fig. X. Middleware Request Processing Pipeline of the WanderLust Platform

Fig. 2. Express middleware pipeline execution order.

A critical ordering constraint exists: the Stripe webhook route must be registered before the JSON body parser because Stripe's signature verification requires the raw request body. Parsing it as JSON first would invalidate the HMAC signature

D. Real-Time Communication Architecture

Socket.IO is integrated as a layer on top of the Node.js HTTP server, enabling three categories of real-time communication:

1) AI Chatbot Communication: The ai_message event triggers the SmartChatbot.handleMessage() function, which processes the user's message through OpenAI's GPT-4o-mini with function calling support. The AI can autonomously decide to call one of four database query functions (search_listings, search_vehicles, search_dhabas, get_user_bookings), execute the query against MongoDB, and return results to the GPT model for natural language response generation. The response is emitted back via the ai_response event.

2) Host–Guest Messaging: Direct messaging between hosts and guests is facilitated through Socket.IO rooms. Each conversation is identified by a unique room ID, and messages are persisted in the Conversation model with a maximum of 20 messages per thread to prevent document bloat.

3) Real-Time Notifications: The notification event enables instant delivery of booking alerts, review notifications, and follow notifications to online users. A connectedUsers Map maintains the mapping between user IDs and socket IDs, with authentication validated using MongoDB ObjectId format regex (/^[a-fA-F0-9]{24}$/).

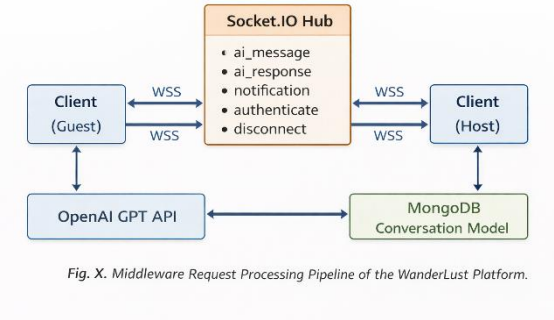

Fig. X. Middleware Request Processing Pipeline of the WanderLust Platform.

Fig. 3. Socket.IO real-time communication architecture.

E. Deployment Architecture

The application is deployed on Render (Platform-as-a-Service), a cloud hosting provider that supports automatic Git-based deployments. The deployment topology is as follows:

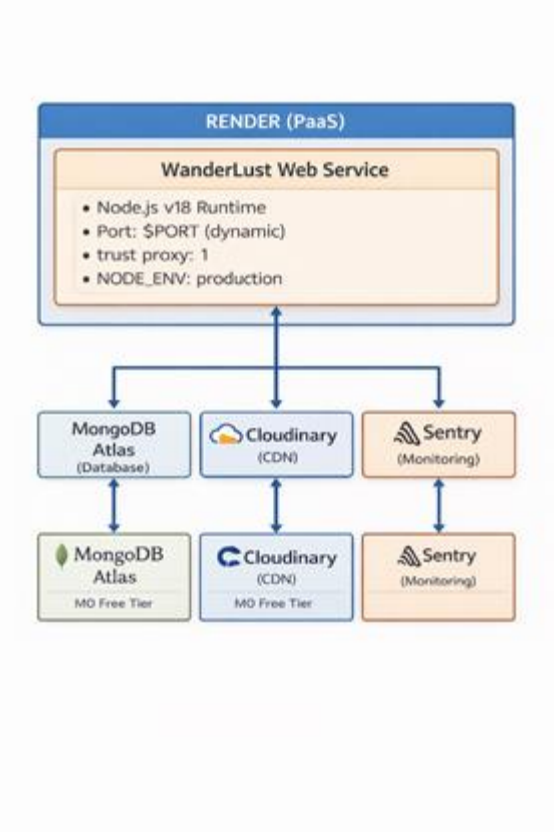

Fig. 4. Deployment architecture on Render with external services.

Key deployment configurations include:

- Session persistence: MongoDB-backed sessions via connect-mongo with a 7-day TTL and touchAfter: 24h to minimize database writes, with automatic fallback to memorystore if the MongoDB session store encounters errors.
- Static asset caching: 1-year immutable Cache-Control headers on JS, CSS, and image assets in production, with ETag and Last-Modified validation.
- Proxy trust: app.set('trust proxy', 1) is configured to ensure correct client IP resolution behind Render's reverse proxy, which is critical for rate limiting accuracy and session cookie secure flag behavior.
- Android deployment: The Capacitor configuration (capacitor.config.json) points the Android WebView to the live Render URL, enabling the Android APK to access the

full server-rendered application without a separate API backend.

The following technologies are proposed for the development of the vacation rental booking system [3].

- Frontend: React.js is used for creating a dynamic and responsive user interface [6].
- Backend: Node.js with Express.js is chosen for building a scalable and efficient server-side application [3].
- Database: MongoDB provides flexible data storage and retrieval for property and user information [2].
- Cloud Platform: AWS or Google Cloud is utilized for hosting and ensuring system scalability [7].
- Payment Gateway: Stripe or PayPal is integrated for secure online payment processing [4].

IV. TECHNOLOGY STACK

Layer	Technology	Version	Purpose
Runtime	Node.js	18.x	Server-side JavaScript runtime
Framework	Express.js	4.18.x	HTTP server & routing
Database	MongoDB Atlas	—	NoSQL document database
ODM	Mongoose	8.x	Object-Document Mapping
Templating	EJS + ejs-mate	3.x	Server-side HTML rendering
Authentication	Passport.js	0.7.x	Local + Google OAuth 2.0
Real-time	Socket.IO	4.8.x	WebSocket bi-directional chat
Payments	Stripe + Razorpay	19.x / 2.9.x	International + Indian payments
Maps	Mapbox SDK	0.16.x	Geocoding & geospatial search
Media	Cloudinary + Multer	1.41.x	Image upload & CDN delivery
AI	OpenAI GPT	4.x SDK	Trip planning & AI assistant
Mobile	Capacitor	8.x	Android/iOS native bridge
Security	Helmet, csurf, hpp	Latest	HTTP headers, CSRF, HPP
Logging	Winston	3.11.x	Structured application logging
Monitoring	Sentry	7.x	Error tracking & performance
Email	Nodemailer / Brevo	7.x	Transactional emails & OTP
Testing	Jest + Supertest	29.x	Unit & integration testing
PDF	PDFKit	0.17.x	Booking confirmation PDFs
QR Codes	qrcode	1.5.x	Booking QR generation
OTP	Speakeasy	2.x	Two-factor authentication
Push	web-push	3.6.x	Browser push notifications

V. IMPLEMENTATION DETAILS

A. Multi-Category Booking System

WanderLust supports three distinct booking categories, each with dedicated models, routes, controllers, and views:

1) Accommodation Listings — Properties include type (Apartment, Villa, Cabin, etc.), capacity, amenities, pricing variations, and calendar-based availability. A compound geospatial index enables proximity search.

2) Vehicle Rentals — Vehicles support per-day and per-hour pricing, security deposits, feature lists, and time-slot availability management.

3) Dhaba (Dining) Reservations — Local restaurants listing specialties, operating hours, vegetarian/vegan flags, and table reservation slots.

All three booking types converge on a unified Booking model, differentiated by a type field (listing | vehicle | dhaba), enabling a single booking history view for users.

```
javascript
// Unified Booking Schema (simplified)
const bookingSchema = new Schema({
  listing:  { type: ObjectId, ref: "Listing" },
  vehicle:  { type: ObjectId, ref: "Vehicle" },
  dhaba:    { type: ObjectId, ref: "Dhaba" },
  user:     { type: ObjectId, ref: "User", required: true },
  type:     { type: String, enum: ['listing','vehicle','dhaba'] },
  paymentMethod: String, // 'Stripe' | 'Razorpay' | 'Wallet' | 'UPI'
  paymentStatus: { type: String, enum: ['Pending','Paid','Failed','Refunded'] },
  status:   { type: String, enum: ['Pending','Confirmed','Cancelled'] }
});
```

B. Authentication & Authorization

Authentication is implemented via Passport.js with two strategies:

- Local Strategy (passport-local-mongoose): Email/password with bcryptjs hashing.
- Google OAuth 2.0 (passport-google-oauth20): Social sign-in via Google.

Authorization is enforced through custom middleware:

Middleware	Purpose
isLoggedIn	Verifies active session; redirects or returns 401 JSON for API requests
isOwner	Confirms the current user owns the resource being modified
isReviewAuthor	Validates review authorship before deletion
isAdmin	Restricts admin panel access to the designated super-admin email
isEmailVerified	Blocks actions until OTP email verification is complete

Role-based access control (RBAC) is enforced via a role field on the User model with values: user, host, admin.

C. Price Calculation Engine

A dedicated utility (utils/price-calculator.js) implements the pricing logic:

```
javascript
function calculateBookingPrice(basePrice, nights, guests = 1) {
  const subtotal     = basePrice * nights;
  const cleaningFee  = Math.max(Math.round(basePrice * 0.1), 500);
  const serviceFee   = Math.round(subtotal * 0.05);
  const extraGuestFee = guests > 2 ? (guests - 2) * 200 * nights : 0;
  const totalBeforeTax = subtotal + cleaningFee + serviceFee + extraGuestFee;
  const tax   = Math.round(totalBeforeTax * 0.12); // 12% GST
  const total = totalBeforeTax + tax;
  return { subtotal, cleaningFee, serviceFee, extraGuestFee, tax, total };
}
```

The formula applies a 10% cleaning fee (min ₹500), 5% service fee, ₹200/night per extra guest beyond 2, and 12% GST, compliant with Indian tax regulations.

D. Payment Gateway Integration

Two payment gateways were integrated to maximize coverage:

Stripe handles international card payments via checkout sessions. Webhook endpoints (/webhook) listen for payment_intent.succeeded and checkout.session.completed events to update booking status asynchronously.

Razorpay supports Indian payment methods including UPI, net banking, and cards. The razorpayController.js creates orders, verifies signatures using HMAC-SHA256, and confirms bookings upon success.

Wallet System: Users maintain an in-app wallet balance (walletBalance on User model) that can be used for instant payments without gateway fees.

E. AI Trip Planner

The AI trip planning module (routes/tripPlanner.js) integrates the OpenAI GPT API to generate personalized travel itineraries. Users input destination, duration, budget, and preferences; the system constructs a structured prompt and returns a day-by-day itinerary in JSON format, rendered dynamically on the frontend.

An AI assistant chatbot (public/js/aiAssistant.js) provides conversational support for property queries, booking guidance, and travel recommendations.

F. Real-Time Chat

Socket.IO powers the bidirectional messaging system between hosts and guests. Conversation threads are stored in a MongoDB Conversation model. The server manages room-based event emission, ensuring messages are delivered only to participants of a specific conversation.

G. Geospatial Search & Maps

Mapbox GL JS renders interactive maps on listing detail pages. The Mapbox SDK's geocodingClient.forwardGeocode() converts user-submitted addresses into GeoJSON Point coordinates, stored in the geometry field of each listing. A 2dsphere index enables MongoDB's $near operator for proximity-based queries.

javascript

```
// Geospatial Index on Listing Schema
listingSchema.index({ geometry: '2dsphere' });
```

H. Image Management

Cloudinary handles all media uploads. The multer-storage-cloudinary package streams uploaded files directly to Cloudinary, avoiding local disk I/O. Each listing supports a primary image and a multi-image gallery with captions and primary-flag designation.

I. Mobile Application

The Capacitor framework wraps the web application into a native Android container. The capacitor. config.json points to the live server URL, allowing the Android WebView to access the full server-rendered application. This "hybrid web-native" approach enables Play Store distribution without requiring a separate React Native or Flutter codebase.

J. Notification Systems

Three notification channels are implemented:

- Email (Nodemailer/Brevo): Booking confirmations, OTP codes, and host alerts.
- Browser Push Notifications (web-push + VAPID keys): Real-time alerts for new bookings.
- In-App Notifications: Stored in the Notification model, displayed via the notification panel.

K. Social Features

The platform implements a social layer inspired by travel community apps:

- Wishlists: Users can save listings, vehicles, and dhabas to category-specific wishlists.
- Travel Journal: Published blog-style posts (Journal model) with social sharing.

- Follow System: Users can follow/unfollow each other, with activity feeds.
- Review & Rating: Reviews with star ratings on all three booking categories.

VI. DATABASE DESIGN

A. Schema Overview

Wander Lust uses 13 Mongoose models:

Model	Purpose
User	Authentication, profile, roles, wallet
Listing	Stay properties with geospatial data
Vehicle	Rental vehicles with pricing
Dhaba	Dining venues
Booking	Unified booking record
Review	User reviews for all categories
Conversation	Chat thread metadata
Notification	In-app notification records
Journal	Travel blog posts
SavedTrip	User-saved itineraries
Subscriber	Newsletter subscribers
OTP	Email verification tokens

B. Key Design Decisions

Referential Integrity: Mongoose post('findOneAndDelete') middleware cascades deletions (e.g., deleting a listing also deletes its reviews and bookings).

Performance Indexing: Compound and single-field indexes were strategically applied:

javascript

```
// Listing Model Indexes
listingSchema.index({ title: 'text', location: 'text', country: 'text' }); // Full-text search
listingSchema.index({ propertyType: 1, price: 1, guests: 1 }); // Compound filter
listingSchema.index({ geometry: '2dsphere' }); // Geospatial
listingSchema.index({ unavailableDates: 1 }); // Availability queries
// Booking Model Indexes
bookingSchema.index({ user: 1, createdAt: -1 }); // User history
bookingSchema.index({ startDate: 1, endDate: 1 }); // Date range
```

Schema Flexibility: The propertyType enum and amenities array allow rich property differentiation without schema migration overhead.

VII. SECURITY IMPLEMENTATION

Security was implemented as a multi-layered defense strategy:

Threat	Countermeasure	Package
XSS Attacks	Content Security Policy, input sanitization	helmet, xss-clean
CSRF	Token-based CSRF on all state-changing requests	csurf
SQL/NoSQL Injection	Query sanitization	express-mongo-sanitize
Brute Force	Rate limiting on auth & booking endpoints	express-rate-limit
HTTP Parameter Pollution	HPP middleware	hpp
Session Hijacking	Secure, httpOnly, sameSite cookies	express-session
Password Storage	bcryptjs hashing (salt rounds: 12)	passport-local-mongoose
Unauthorized Access	Role-based middleware guards	Custom middleware
Data Exposure	getPublicProfile() strips sensitive fields before API response	Custom method

Rate limiting configuration:

- Authentication routes: 10 requests per 15 minutes
- Booking initiation: 5 requests per 15 minutes
- General API: 100 requests per 15 minutes.

VIII. TESTING

A. Testing Framework

Testing was implemented using Jest as the test runner and Supertest for HTTP integration testing. mongodb-memory-server provides an in-memory MongoDB instance for isolated test execution without affecting the production database.

B. Test Structure

```
tests/
├── unit/          # Pure function unit tests
│   ├── priceCalculator.test.js
│   └── middleware.test.js
└── integration/   # API endpoint tests
    ├── listings.test.js
    ├── bookings.test.js
    └── users.test.js
```

C. Test Coverage

Unit tests validated the price calculator's GST computation, service fee rounding, and extra-guest fee thresholds. Integration tests verified authentication flows, CRUD operations, and booking state transitions. Jest's --coverage flag was used to track coverage metrics.

IX. PERFORMANCE OPTIMIZATIONS

Optimization	Implementation
HTTP Compression	compression middleware (gzip) on all responses
Asset Minification	Custom scripts/minify-assets.js using terser (JS) and clean-css (CSS)
Node Caching	node-cache for frequently accessed, rarely-changed data
Database Indexing	10+ strategic indexes across primary models
Cloudinary CDN	Images served from Cloudinary's global CDN
Session Storage	connect-mongo for persistent sessions with memorystore fallback

X. RESULTS AND DISCUSSION

A. Feature Completeness

The following table summarizes implemented features against planned objectives:

Feature	Status
Multi-category booking (stay/vehicle/dhaba)	✓ Complete
User authentication (local + Google OAuth)	✓ Complete
Email OTP verification	✓ Complete
AI Trip Planner (OpenAI GPT)	✓ Complete
Real-time chat (Socket.IO)	✓ Complete
Stripe payment integration	✓ Complete
Razorpay payment integration	✓ Complete
Wallet & UPI payment system	✓ Complete
Geospatial map search (Mapbox)	✓ Complete
Cloudinary image gallery	✓ Complete
Android mobile app (Capacitor)	✓ Complete
Social features (wishlist, journal, follow)	✓ Complete
Admin dashboard	✓ Complete
Push notifications (web-push)	✓ Complete
Booking PDF & QR generation	✓ Complete
Security (Helmet, CSRF, rate limiting)	✓ Complete
Error monitoring (Sentry)	✓ Complete

B. Scalability Analysis

The MongoDB Atlas deployment with compound indexes supports horizontal scaling. The application's stateless Express design (session state externalized to MongoDB via connect-mongo)

enables multi-instance deployment behind a load balancer with no architectural changes.

C. Limitations

- The AI trip planner depends on OpenAI API availability and incurs per-request costs.
- The Android app uses a WebView bridge, limiting access to certain native device APIs.
- The current deployment on Render's free tier may experience cold-start latency of 30–60 seconds.

XI. CONCLUSION

This paper presented the design and implementation of WanderLust, a comprehensive multi-category travel booking platform developed as a final-year project. The system successfully integrates accommodation, vehicle rental, and dining reservations into a unified platform enhanced with AI trip planning, real-time communication, dual-gateway payments, and geospatial search capabilities.

The implementation demonstrates that a small development team can produce a production-grade, feature-rich application using modern open-source technologies. The platform's MVC architecture, security-first design, and modular codebase establish a strong foundation for future enhancements including microservices migration, iOS support, and advanced ML-based recommendation engines.

Future work includes implementing server-side pagination for large dataset performance, integrating native Android push notifications via Capacitor's Push plugin, expanding the AI module with multi-modal image understanding, and adding support for group travel coordination features.

REFERENCES

1. N. Bhatt et al., "Scaling Airbnb's distributed data infrastructure," Proc. ACM SIGMOD, 2019.
2. L. Chen, R. Kumar, and S. Patel, "LLM-based travel itinerary generation: A user study," IEEE Trans. Inf. Syst., vol. 12, no. 3, pp. 45–57, 2023.
3. M. Fowler and P. Sadalage, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2012.
4. A. Sharma and B. Gupta, "Performance analysis of MERN stack for real-time web applications," Proc. IEEE ICCC, pp. 112–117, 2022.
5. L. Klein, "Real-time web communication with Socket.IO," J. Web Eng., vol. 8, no. 2, pp. 88–104, 2021.
6. Ionic Framework, "Capacitor: Cross-platform native runtime for web apps," [Online]. Available: https://capacitorjs.com, 2024.
7. Express.js Foundation, "Express.js Documentation," [Online]. Available: https://expressjs.com, 2024.
8. MongoDB Inc., "MongoDB Atlas Documentation," [Online]. Available: https://www.mongodb.com/atlas, 2024.
9. OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.
10. Cloudinary Inc., "Cloudinary Node.js SDK Documentation," [Online]. Available: https://cloudinary.com/documentation, 2024.