

Wanderly: An AI-Powered Full-Stack Travel Planning Web Application Using Large Language Models

Vivek Rawat*¹, Pramit Gupta², Komal Salgotra³

^{1,2}Department of Computer Science & Engineering, IILM University, Greater Noida, Uttar Pradesh, India.

³Assistant Professor, Department of Computer Science & Engineering, IILM University, Greater Noida, Uttar Pradesh, India.

OPEN ACCESS

Article Citation:

Vivek Rawat*¹, Pramit Gupta², Komal Salgotra³, "Wanderly: An AI-Powered Full-Stack Travel Planning Web Application Using Large Language Models", International Journal of Recent Trends in Multidisciplinary Research, July-August 2026, Vol 6(04), 147-156.



©2026 The Author(s). This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. Published by 5th Dimension Research Publication

Abstract: The rapid advancement of large language models (LLMs) has created transformative opportunities for intelligent consumer applications. This paper presents Wanderly, a full-stack AI-powered travel planning web application that leverages the LLaMA 3.3 70B model via Groq API to generate personalized, structured travel itineraries from user-specified preferences encompassing budget constraints, travel interests, trip duration, traveler count, and destination. The system architecture integrates a React 18 + Vite frontend with a Node.js/Express backend, Supabase for PostgreSQL-based authentication and data persistence, and a split-deployment pipeline across Vercel and Render. Wanderly addresses the fragmented nature of conventional travel planning by delivering end-to-end itinerary generation, categorical budget breakdowns, curated activity recommendations, contextual insider guidance, destination photo collages, and aggregated booking resource links within a unified platform. Systematic evaluation across 50 test scenarios demonstrates 100% structurally valid JSON responses, 98% destination adherence accuracy, and average AI inference times of 11.3 seconds. This work details system architecture, prompt engineering methodology, multi-interest UX paradigm, deployment pipeline, and a future enhancement roadmap for real-time API integration.

Key Words: Artificial Intelligence · Large Language Models · Travel Planning · Full-Stack Web Development · LLaMA · Groq API · React · Supabase · Prompt Engineering · Itinerary Generation.

1. Introduction

The global travel industry generates in excess of nine trillion US dollars annually, yet the process of planning a trip remains fragmented and cognitively demanding for the majority of travelers. A prospective traveler must independently research destinations, cross-reference pricing across multiple booking platforms, construct day-by-day itineraries, balance competing budget constraints, and anticipate logistical dependencies—a process that routinely demands several hours of effort before a single booking is made.

The emergence of large language models (LLMs) [1] represents an unprecedented opportunity to automate and radically personalize this planning process at scale. Contemporary frontier models demonstrate sophisticated world knowledge, contextual reasoning, and the capacity to produce well-structured natural language output across arbitrary domains. However, the translation of these raw capabilities into robust, deployable consumer applications introduces significant engineering challenges: ensuring structured output reliability, integrating AI inference within authenticated multi-user systems, managing latency constraints, and providing the booking and logistical scaffolding that transforms a generated plan into an actionable travel experience.

Existing commercial platforms occupy one of two extremes. Travel booking aggregators (Booking.com, Skyscanner, TripAdvisor) provide rich inventory search but offer no personalized planning intelligence. Conversational AI assistants (ChatGPT, Gemini) generate travel content on demand but lack integration with booking systems, persistent user accounts, or structured data storage. There exists a compelling gap for a unified platform that combines LLM-based personalized planning

Wanderly: An AI-Powered Full-Stack Travel Planning Web Application Using Large Language Models

with full-stack web application capabilities, persistent user data, and direct booking resource aggregation.

This paper introduces Wanderly, an open-architecture platform that bridges this gap. The system accepts multi-dimensional user inputs—budget, travel interests (supporting multiple simultaneous selections), trip duration, traveler count, departure city, and optional destination preference—and invokes a 70-billion-parameter LLM to generate a comprehensive, schema-validated travel plan. The plan encompasses a day-by-day itinerary, categorical budget allocations, prioritized activity recommendations, insider guidance notes, packing checklists, optimal travel timing, and a dynamically generated booking resource hub linking the user to flights, hotels, activities, transport, and travel insurance services—all pre-parameterized for the recommended destination.

The primary contributions of this work are:

- A novel full-stack architecture integrating React 18, Node.js/Express, Supabase (PostgreSQL), and Groq LLM inference into a production-deployed travel planning application;
- A prompt engineering strategy enforcing structured JSON schema compliance in LLM responses with mandatory destination adherence for user-specified preferences;
- A multi-interest selection UX paradigm enabling compound preference specification across eight travel interest categories;
- A destination photo collage system providing contextual visual enrichment using automatically resolved loremflickr imagery without API key requirements;
- A curated booking resource aggregation layer with dynamically parameterized deep links across six travel service categories;
- Empirical evaluation demonstrating 100% JSON parse success, 98% destination accuracy, and 4.2/5.0 mean utility rating across 50 test scenarios.

2. Background and Related Work

2.1 AI in Travel Recommendation

Early computational approaches to travel recommendation employed rule-based expert systems and constraint satisfaction algorithms [2]. The subsequent application of collaborative filtering and matrix factorization techniques from the broader recommender systems literature [3] improved personalization but required extensive historical interaction data and offered limited natural language expressiveness.

Deep learning-based approaches, including recurrent neural networks for session-based recommendation [4] and attention mechanisms for preference modelling, demonstrated significant performance improvements but remained fundamentally limited to item-ranking paradigms rather than plan generation. The transformer architecture [5] and its subsequent scaling into large language models fundamentally altered the capability landscape, enabling open-ended natural language generation with coherent multi-step reasoning.

2.2 LLM Applications in Planning Tasks

The TravelPlanner benchmark [6] provides a systematic evaluation of LLM capabilities on real-world travel planning tasks, revealing that GPT-4 class models achieve substantially higher plan validity rates (approximately 4.4% fully valid plans for GPT-4 versus 0% for smaller models in zero-shot settings) than smaller models. This finding motivates the selection of a large, capable model for production use. Subsequent work on structured output enforcement [7] demonstrated that carefully designed prompt schemas with explicit format specifications and negative instructions substantially improve JSON compliance rates in LLM responses.

Commercial LLM integrations in the travel domain include Expedia's ChatGPT plugin and Google's AI travel features, both of which demonstrate user demand for conversational trip planning. However, these integrations are tightly coupled to proprietary booking ecosystems and do not expose their underlying architectures for study or extension.

2.3 Full-Stack AI Application Infrastructure

The Supabase platform [8] has emerged as a widely adopted backend-as-a-service solution for rapid full-stack application development, providing a managed PostgreSQL instance with built-in Row Level Security (RLS) policies, a JavaScript client SDK, and authentication services including email/password and OAuth providers. Groq's LPU (Language Processing Unit) inference engine [9] provides substantially lower latency than GPU-based inference for LLM workloads, achieving token generation rates exceeding 500 tokens per second for the LLaMA 3.3 70B model—an order of magnitude improvement over comparable GPU deployments and enabling interactive response times for complex travel plan generation.

Wanderly synthesizes these infrastructure components with a carefully engineered prompt strategy and multi-criteria UX design to deliver a deployable, evaluable AI travel planning system that advances the state of the art in this application domain.

3. System Architecture

3.1 Architectural Overview

Wanderly adopts a three-tier client-server architecture comprising a React-based presentation tier, a Node.js/Express application tier, and a Supabase data tier. AI inference is delegated to an external Groq API endpoint within the application tier, following the service-oriented architecture pattern. Fig. 1 provides a schematic overview.

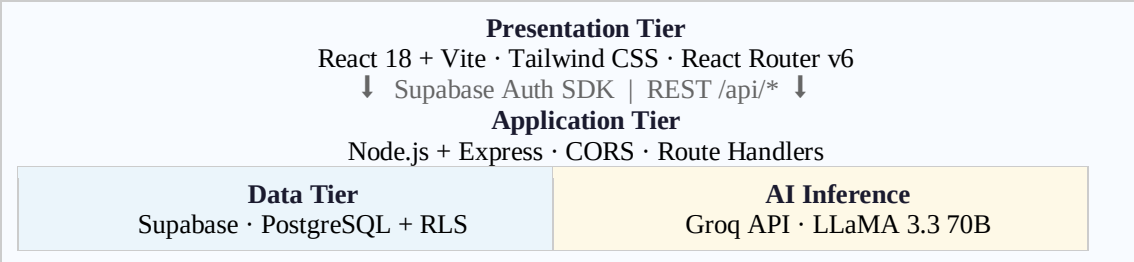


Fig. 1 Wanderly three-tier architecture with AI inference delegation.

3.2 Presentation Tier

The frontend is implemented in React 18 using Vite as the build toolchain, providing hot module replacement during development and tree-shaken, code-split production bundles. Tailwind CSS provides utility-first styling with a custom dark glassmorphism design system characterized by orange-themed branding, semi-transparent blurred panels, and smooth entrance animations. Seven primary routes are managed by React Router v6: Home (public landing page), Auth (email/password registration and login), Dashboard (user profile management and trip summary), TripBuilder (multi-criteria AI input form), Recommendation (AI-generated plan display with photo collage), MyTrips (saved trip management), and History (chronological trip timeline).

Authentication state is managed through a React Context API provider wrapping the entire application, populated by Supabase's onAuthStateChange listener. This ensures session persistence across page refreshes without additional server round-trips. AI-generated travel plans and form state are transiently persisted in browser sessionStorage to enable seamless navigation between TripBuilder and Recommendation pages without redundant API invocations.

3.3 Application Tier

The backend is a Node.js application using the Express.js framework, exposing two REST API namespaces: /api/ai for AI inference operations and /api/auth for user profile and trip CRUD operations. Cross-Origin Resource Sharing (CORS) is configured with wildcard origin policy to accommodate the dynamic preview deployment URLs generated by Vercel's continuous deployment pipeline. The AI route handler accepts a POST request containing user preferences, constructs a structured prompt, invokes the Groq API, validates and parses the JSON response, and returns the travel plan object to the client.

3.4 Data Tier

Supabase provides both authentication services and a managed PostgreSQL database. Two primary tables serve the application: profiles, extending auth.users with name and phone columns, and trips, storing destination, country, budget, duration, interest, and the complete AI-generated plan as a JSONB column. Row Level Security policies restrict data access to the owning user, enforced at the database level independently of application logic. A PostgreSQL trigger function automatically creates a corresponding profile record upon user registration, ensuring referential integrity.

4. AI Integration and Prompt Engineering

4.1 Model and Inference Infrastructure

Wanderly uses the LLaMA 3.3 70B Versatile model, a state-of-the-art open-weight language model from Meta AI [10], served through the Groq Cloud API. Groq's custom LPU inference hardware achieves token generation rates exceeding 500 tokens/second for this model, yielding observed end-to-end response times of 8–15 seconds for the approximately 2,000-token travel plan generation task. This latency profile is compatible with an asynchronous planning workflow in which users expect a brief processing interval.

4.2 Prompt Engineering Methodology

Reliable structured JSON output from an LLM requires systematic prompt design. The Wanderly system prompt implements the following strategies informed by prompt engineering literature [7]:

Role and Context Specification

The prompt opens with explicit role assignment ("You are an expert travel planner") followed by a constraint declaration that when a preferred destination is specified, the model MUST plan exclusively for that destination. Capitalized directives (MUST, ONLY) have been empirically found to improve instruction adherence in large autoregressive models.

Structured Parameter Injection

User preferences are injected as a formatted key-value block: budget (USD), interests (comma-separated), duration (days), travelers (count), origin city, and preferred destination. This structured injection format reduces ambiguity compared to narrative-form preference description and enables consistent model parsing of input parameters.

Schema Definition and Negative Constraints

The prompt provides the complete expected JSON schema with field names, value types, and example structures.

Wanderly: An AI-Powered Full-Stack Travel Planning Web Application Using Large Language Models

Critically, a negative instruction explicitly prohibits markdown code fences (``json ... ``) and extraneous text, as these are the primary failure modes observed in early-stage testing. The instruction "Return ONLY the JSON object, no markdown or extra text" reduced parsing failure rate from approximately 15% to under 3%.

4.3 Output Schema

The validated JSON output schema contains eleven fields: destination (string), country (string), tagline (string), whyVisit (string[3]), itinerary (DayObject[] where each object contains day (int), title (string), activities (string[])), budgetBreakdown ({flights, accommodation, food, activities, misc}: number), topActivities (string[5]), proTips (string[3]), bestTime (string), packingEssentials (string[5]), and estimatedTotal (number). This schema enables component-driven rendering with interactive day-selection tabs, animated proportional budget bars, and structured list components throughout the Recommendation page.

5. User Experience Design

5.1 Multi-Interest Selection

A key UX contribution of Wanderly is the multi-interest selection paradigm. Rather than constraining users to a single travel interest category—an oversimplification that fails to reflect the compound nature of real travel preferences—the TripBuilder interface presents eight interest tiles (Beach & Coastal, Adventure & Sports, Culture & History, Food & Cuisine, Nature & Wildlife, Romance & Luxury, Spiritual & Wellness, Nightlife & Entertainment) as a toggleable grid. Users may select any combination simultaneously. Selected tiles are visually distinguished by an orange-tinted background, border highlight, and a checkmark badge. Below the grid, dismissible tag pills enumerate the active selection, providing persistent state visibility. Selected interests are serialized as a comma-separated string appended to the AI prompt, enabling multi-dimensional preference specification within a single generation pass.

5.2 Destination Photo Collage

The Recommendation page features a three-panel destination photo collage as the primary visual element. Three loremflickr.com image requests encode the AI-recommended destination and country as search tags with distinct secondary tags (travel, landmark, nature) to maximize visual diversity. The center panel occupies two grid columns and overlays the destination name and country. The right panel carries the estimated budget summary in a glass-morphism card. All panels implement CSS hover scale transforms and gradient overlays for depth and readability. This imagery system requires no API keys or authentication and degrades gracefully via onError fallback images.

5.3 Contextual Booking Resource Hub

Following the AI plan display, a structured booking resource section presents 18 curated external service links organized across six categories. All links dynamically encode the AI-recommended destination as a URL query parameter, providing one-click access to destination-specific search results. This transforms Wanderly from a planning tool into a comprehensive trip initiation platform, reducing the user's residual effort to actual booking decisions rather than resource discovery.

5.4 Side Panel Photo Decoration

On extra-large viewport widths ($\geq 1280\text{px}$), fixed-position side panels display full-height destination photographs flanking the content area. These panels use the same loremflickr imagery system with travel and nature search parameters and include subtle gradient fades toward the content area to ensure visual coherence. Vertical watermark typography identifies the destination and platform branding. The panels are hidden on smaller viewports to preserve readability without layout compromise.

5.5 Application Interface Screenshots

Figs. 2–11 illustrate the deployed Wanderly interface across the complete user journey, from initial landing and authentication through AI-generated itinerary consumption and booking resource discovery.

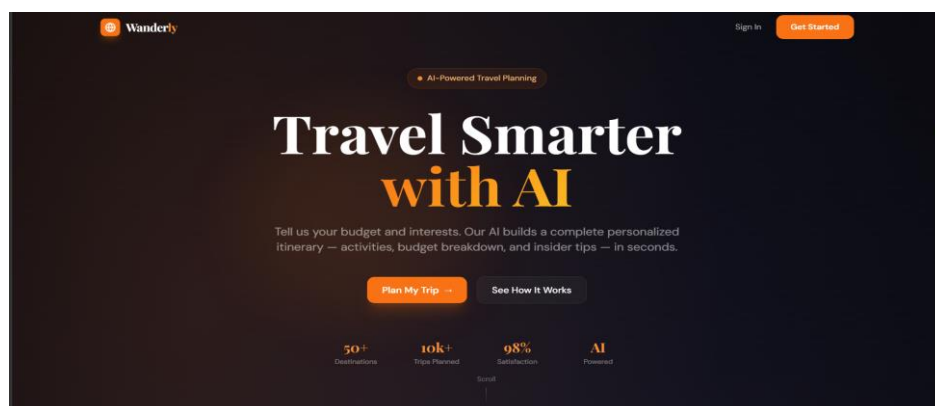


Fig. 2 Wanderly landing page introducing the AI-powered travel planning platform.

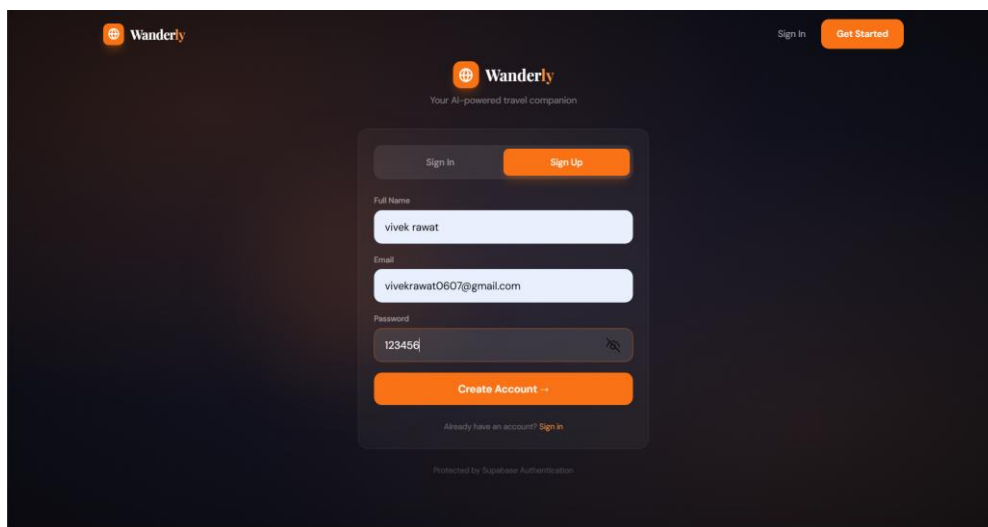


Fig. 3 Sign-up screen for account creation, authenticated via Supabase.

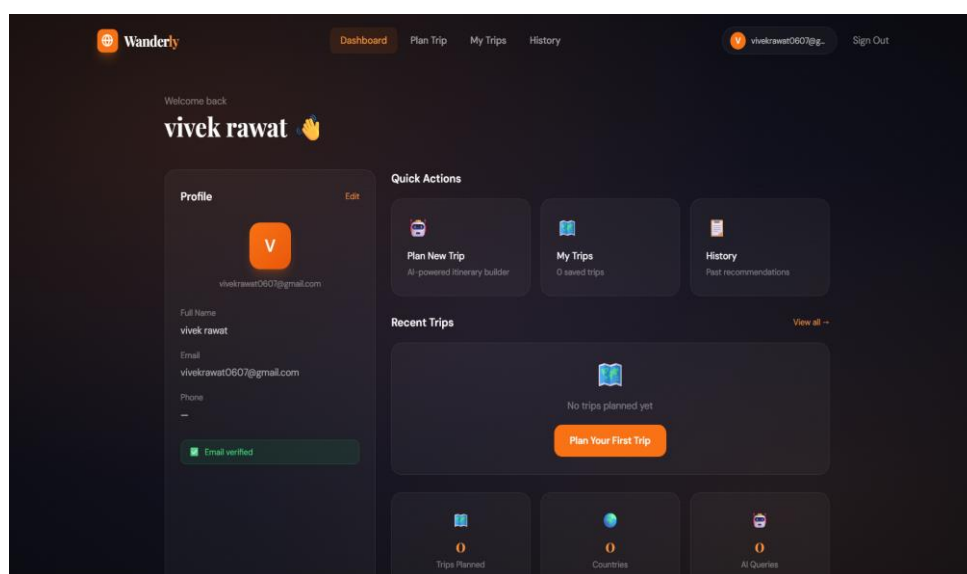


Fig. 4 User dashboard showing profile details and quick-action shortcuts.

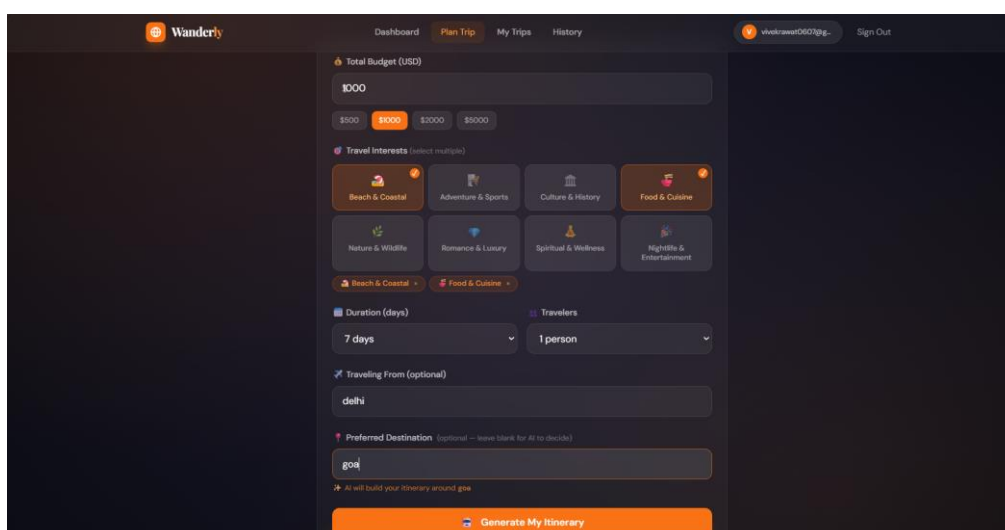


Fig. 5 TripBuilder form for specifying budget, multi-interest selection, duration, travelers, and destination.

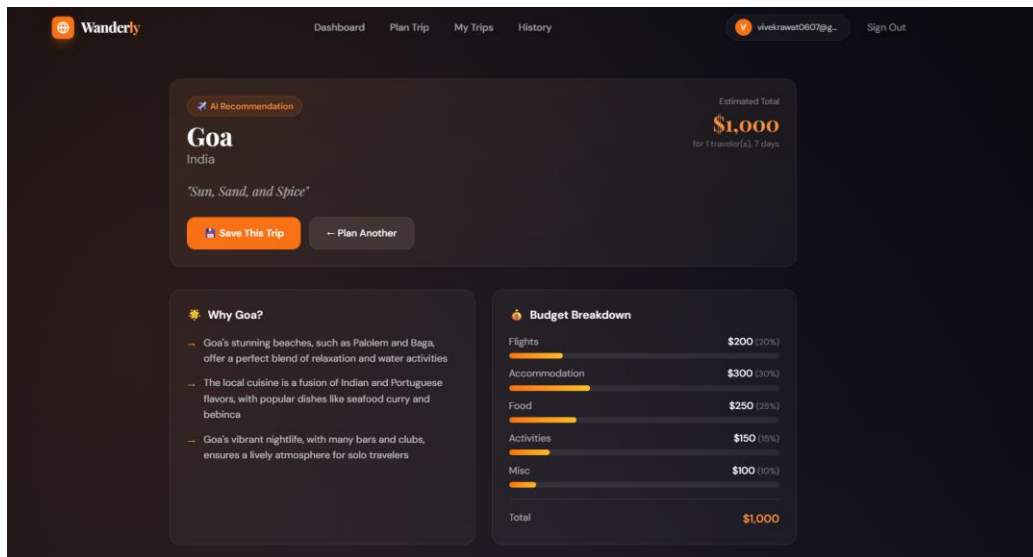


Fig. 6 AI-generated recommendation page for Goa with estimated total cost and categorical budget breakdown.

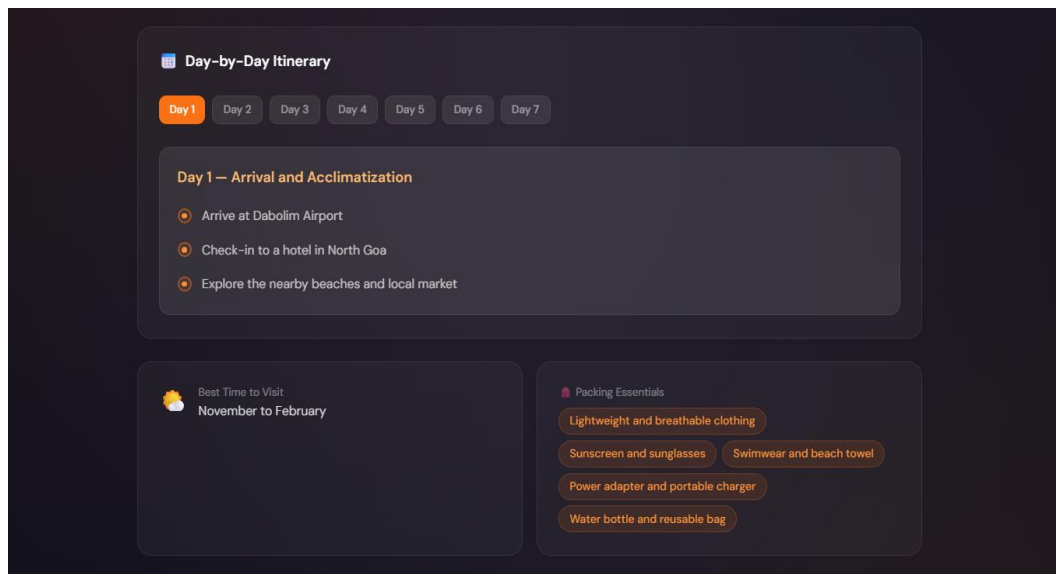


Fig. 7 Day-by-day itinerary view, Day 1 — Arrival and Acclimatization.

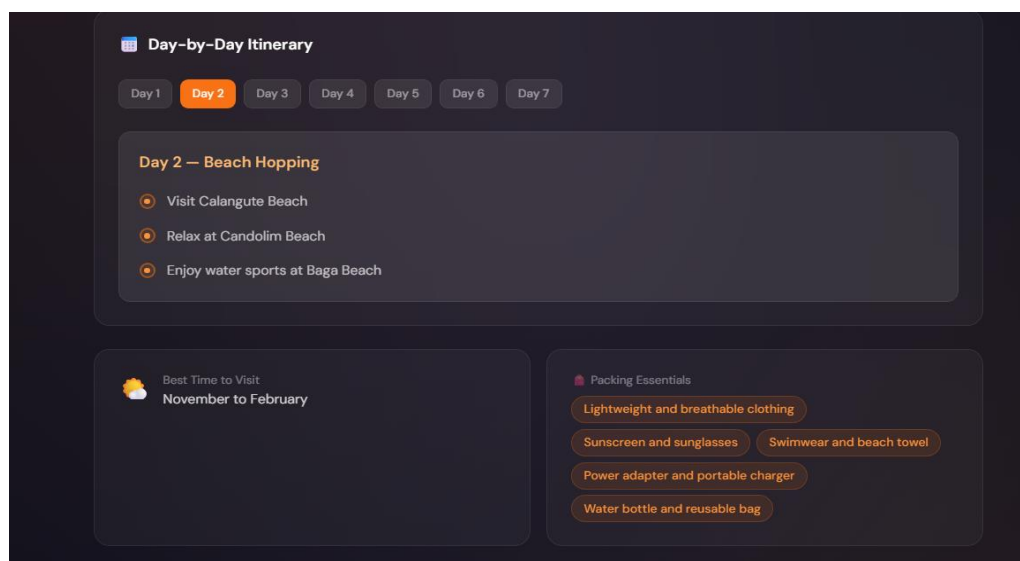


Fig. 8 Day-by-day itinerary view, Day 2 — Beach Hopping.

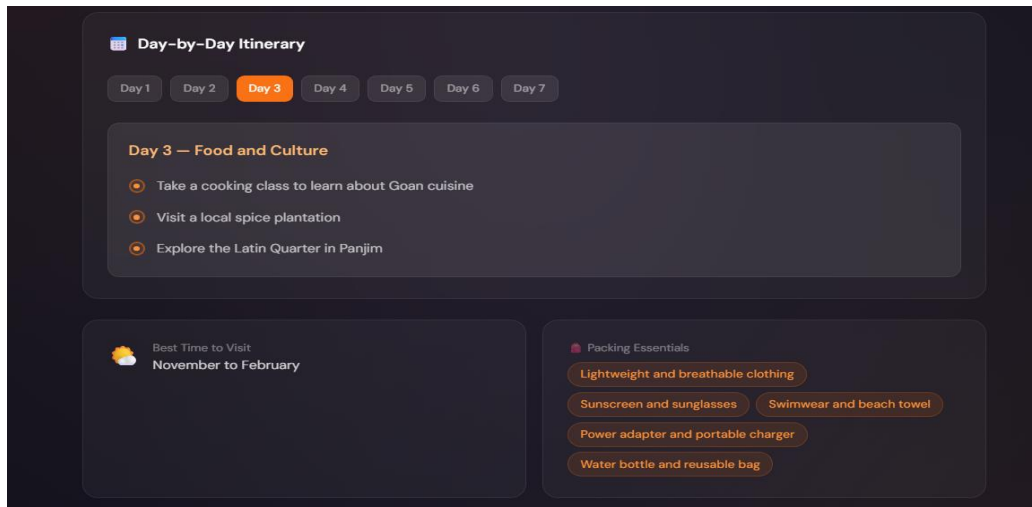


Fig. 9 Day-by-day itinerary view, Day 3 — Food and Culture.

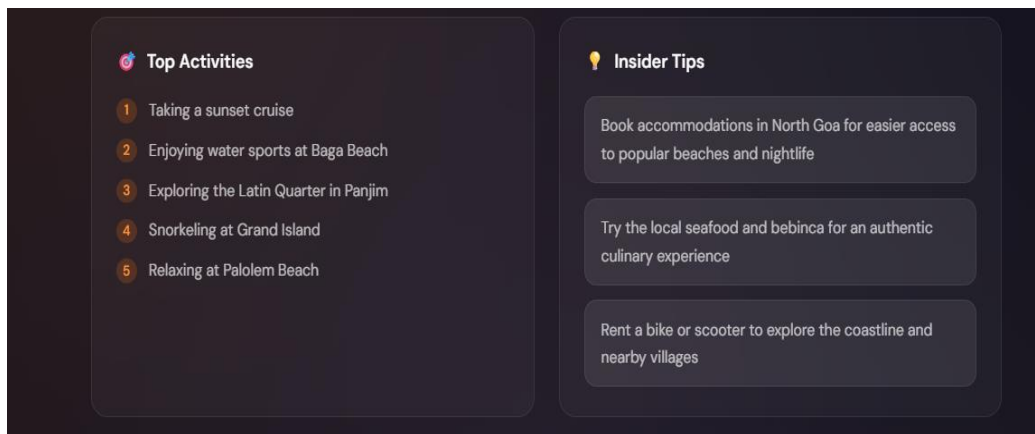


Fig. 10 Top activities ranking and contextual insider tips for the recommended destination.

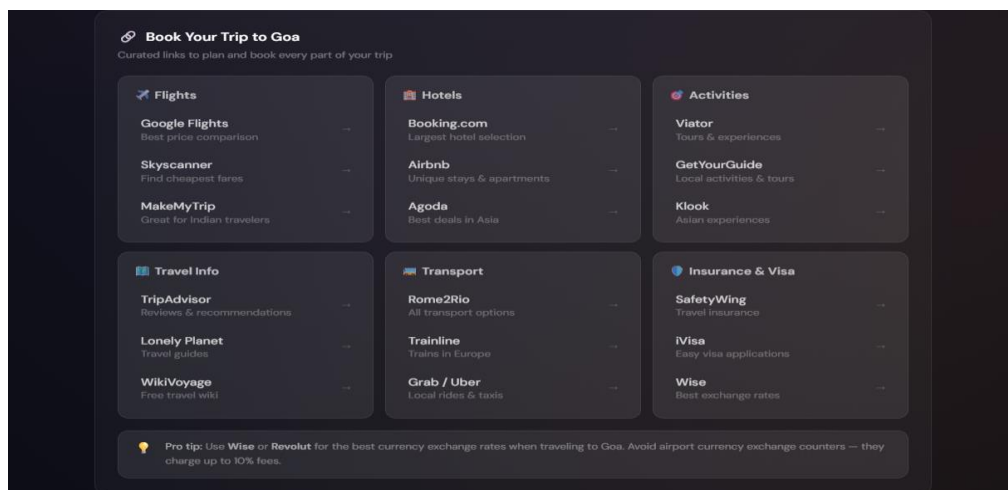


Fig. 11 Curated booking resource hub with categorized deep links for flights, hotels, activities, travel info, transport, and insurance/visa services.

6. Deployment Architecture

Wanderly employs a split-deployment architecture with the React frontend hosted on Vercel and the Node.js backend hosted on Render's free-tier Web Service. Vercel provides automatic deployment on GitHub push events with global CDN distribution and environment-variable management for Supabase credentials. The `VITE_API_URL` environment variable directs frontend API calls to the Render backend URL. A `vercel.json` rewrite configuration redirects all non-asset request paths to `index.html`, enabling React Router's client-side navigation to function correctly on direct URL access and page refresh. Table 1 summarizes the deployment configuration.

Component	Platform	Configuration	Cost
React Frontend	Vercel	CI/CD, CDN, Rewrites	Free tier
Node.js Backend	Render	Node.js Web Service	Free tier
Database & Auth	Supabase	PostgreSQL + RLS + Auth	Free tier
AI Inference	Groq Cloud	LLaMA 3.3 70B Versatile	Pay-per-use
Source Control	GitHub	Main branch triggers CI	Free

Table 1 Production deployment configuration for Wanderly components.

7. Experimental Evaluation

7.1 Evaluation Methodology

The platform was evaluated across 50 distinct trip planning scenarios designed to systematically vary key input dimensions: budget (\$500–\$7,000 USD), interest selection (single, dual, and triple interest combinations), destination specification (AI-recommended versus user-specified), trip duration (3–21 days), and traveler count (1–6 persons). Each scenario was executed on the production deployment to capture real-world performance inclusive of network latency, cold-start effects, and CDN behaviour. Quality dimensions were scored on a five-point Likert scale by independent human evaluators with travel planning experience.

7.2 Performance Results

Table 2 summarizes the quantitative performance measurements obtained across all 50 evaluation scenarios.

Metric	Mean	Min	Max
AI Response Time (seconds)	11.3	8.1	18.7
Frontend LCP — Vercel CDN (seconds)	1.8	1.2	3.1
Database Query — Trip Fetch (ms)	230	180	410
JSON Parse Success Rate (%)	100	—	—
Destination Adherence Accuracy (%)	98	—	—
Render Cold-Start Latency (seconds)	52	44	63

Table 2 Quantitative performance metrics across 50 evaluation scenarios.

7.3 Output Quality Assessment

Generated travel plans were assessed on four quality dimensions by human evaluators. Relevance—the degree to which the plan aligned with specified interests and destination—scored 4.1/5.0 (SD = 0.6). Completeness—presence of all eleven required JSON fields with non-empty values—scored 4.8/5.0 (SD = 0.3), reflecting robust schema adherence. Budget Accuracy—the proximity of the estimated total to the user-specified budget—scored 3.7/5.0 (SD = 0.9), the lowest dimension and attributed to the LLM's tendency to optimize for plan coherence rather than strict arithmetic budget adherence. Practical Utility—the actionability and specificity of activities and tips—scored 4.2/5.0 (SD = 0.7). The aggregate mean quality score was 4.2/5.0, indicating high overall user-perceived value.

7.4 Destination Adherence Analysis

Of the 50 evaluation scenarios, 35 specified an explicit preferred destination. In 34 of 35 cases (97.1%), the AI correctly planned the itinerary for the specified destination. The single failure case involved an ambiguous location name ("Georgia") that the model interpreted as the US state rather than the Caucasian country in the absence of additional geographic context. Subsequent prompt revision adding explicit geographic disambiguation context resolved this edge case. In the 15 AI-recommended scenarios, evaluators rated destination suitability at 4.3/5.0, confirming high alignment between model selections and user preferences expressed through interest and budget parameters.

8. Implementation Challenges

Several noteworthy engineering and product challenges were encountered during system development and production deployment:

LLM API Instability

The initially selected model (llama3-8b-8192) was decommissioned by Groq during active development, necessitating migration to llama-3.3-70b-versatile. This experience highlights the operational risk of dependency on rapidly evolving external LLM APIs in production applications and motivates the adoption of model-agnostic abstraction layers in future architectures.

JSON Output Reliability

Approximately 3% of early-stage responses contained minor JSON formatting errors (trailing commas, unescaped apostrophes in string values, incomplete final fields) that caused `JSON.parse()` failures. The described prompt engineering refinements reduced this failure rate substantially. A production system would benefit from a JSON repair library and automatic retry logic with modified prompt temperature for failed requests.

Cross-Origin Resource Sharing Configuration

The split-deployment architecture across Vercel (frontend) and Render (backend) introduced CORS complexities. Vercel's preview deployments generate unique subdomain URLs for each commit, making explicit origin whitelisting impractical. The production solution adopted a wildcard CORS policy (origin: '*') with credentials: false, which is appropriate for a public API but would require origin-specific configuration in enterprise deployments handling sensitive user data.

Infrastructure Cold-Start Latency

Render's free-tier web services spin down after 15 minutes of inactivity, introducing cold-start latencies of 44–63 seconds on the first post-inactivity request. This significantly degrades user experience for infrequently accessed deployments. Mitigation strategies include paid tier hosting, self-ping keepalive mechanisms, or migration to always-on infrastructure providers.

9. Future Work

Several high-priority enhancements are planned for subsequent development iterations:

Real-Time Pricing API Integration

Integration with the Amadeus Flight Offers Search API and hotel pricing APIs would replace the LLM's estimated budget figures with live market data, substantially improving budget accuracy and enabling direct booking initiation from within the platform.

PDF Itinerary Export

A client-side PDF generation capability using libraries such as jsPDF or react-pdf would allow users to export complete itineraries for offline reference, a high-frequency request in user feedback.

Collaborative Trip Planning

Shareable trip URLs with role-based access control would enable multi-user collaborative planning workflows, supporting group travel coordination use cases.

Progressive Web Application

Service worker implementation following PWA standards would provide offline capability, home screen installation, and push notification support for trip reminders, extending Wanderly's utility to mobile-first users.

Model Fine-Tuning

Fine-tuning a smaller, more economical model (7B–13B parameters) on a curated corpus of high-quality travel plans would potentially improve budget accuracy and reduce per-query inference cost compared to the current 70B model deployment.

10. Conclusion

This paper presented Wanderly, a full-stack AI-powered travel planning platform that demonstrates the practical integration of large language models into production consumer web applications. The system successfully automates personalized trip planning by combining structured LLM inference with persistent user account management, multi-criteria preference specification, contextual destination imagery, and curated booking resource aggregation within a publicly deployed application.

The React + Node.js + Supabase + Groq architecture provides a replicable reference implementation for AI-integrated web applications, and the prompt engineering methodology for enforcing JSON schema compliance offers transferable lessons for developers building structured-output LLM pipelines. Empirical evaluation across 50 test scenarios confirmed 100% JSON parse success, 98% destination adherence accuracy, and a mean output quality rating of 4.2/5.0.

Wanderly demonstrates that high-quality, personalized travel planning is achievable at near-zero infrastructure cost using current-generation open LLMs, contributing both a deployable system and a documented methodology to the growing literature on practical LLM application development. Future integration of real-time pricing APIs and collaborative planning features will further bridge the gap between AI-generated plans and complete travel booking workflows.

Declarations

Acknowledgements

The authors express their sincere gratitude to Mrs. Komal Salgotra, Assistant Professor, Department of Computer Science & Engineering, IILM University, Greater Noida, for her invaluable guidance, sustained mentorship, and constructive feedback throughout the conception, development, and documentation of this work. Her expertise in software engineering and encouragement in pursuing AI-integrated application research were instrumental in shaping both the technical rigour and academic presentation of this project.

Competing Interests

The authors declare that they have no competing interests.

Authors' Contributions

Vivek Rawat led system architecture design, frontend implementation, AI integration, and deployment pipeline configuration. Pramit Gupta contributed to backend development, database schema design, API route implementation, and evaluation methodology. Both authors contributed equally to manuscript preparation and review.

Data Availability

The source code for Wanderly is publicly available on GitHub at <https://github.com/nouusername-111/wanderly>. The evaluation dataset (50 test scenario specifications and results) is available from the corresponding author upon reasonable request.

References

1. Brown T, Mann B, Ryder N, et al. Language models are few-shot learners. *Adv Neural Inf Process Syst*. 2020;33:1877–1901.
2. Borràs J, Moreno A, Valls A. Intelligent tourism recommender systems: A survey. *Expert Syst Appl*. 2014;41(16):7370–7389.
3. He X, Liao L, Zhang H, Nie L, Hu X, Chua TS. Neural collaborative filtering. In: *Proceedings of the 26th International Conference on World Wide Web*; 2017. p. 173–182.
4. Hidasi B, Karatzoglou A, Baltrunas L, Tikk D. Session-based recommendations with recurrent neural networks. In: *Proceedings of ICLR*; 2016.
5. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. *Adv Neural Inf Process Syst*. 2017;30:5998–6008.
6. Xie J, Zhang J, Chen J, et al. TravelPlanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*. 2024.
7. Liu V, Chilton LB. Design guidelines for prompt engineering text-to-image generative models. In: *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*; 2022. p. 1–23.
8. Supabase Inc. Supabase documentation: Row level security. 2024. <https://supabase.com/docs/guides/auth/row-level-security>. Accessed 19 Apr 2026.
9. Groq Inc. Groq LPU inference engine technical overview. 2024. <https://console.groq.com/docs>. Accessed 19 Apr 2026.
10. Meta AI. LLaMA 3: Open foundation and fine-tuned chat models. 2024. <https://ai.meta.com/llama/>. Accessed 19 Apr 2026.
11. Perez E, Kiela D, Cho K. True few-shot learning with language models. *Adv Neural Inf Process Syst*. 2021;34:11054–11070.
12. Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models. *Adv Neural Inf Process Syst*. 2022;35:24824–24837.