Travel Planner: LLMs and Services for Smart Itinerary Generation

Sundaram, Sambhav Mishra, Aggimalla Abhishek, Nenavath Likhith Naik, Snehalatha S H, Animesh Chaturvedi
Department of Data Science and Artificial Intelligence
Indian Institute of Information Technology, Dharwad
Dharwad, Karnataka, India
{23bds060, 23bds050, 23bds004, 23bds037, snehalatha.24phddi12}@iiitdwd.ac.in
animesh@iiitdwd.ac.in

Abstract

Travel planning traditionally requires users to navigate across multiple platforms—airlines, hotels, local attractions, and weather services—resulting in fragmented and time-consuming experiences. This work presents a Large Language Model (LLM)-driven Travel Planner, a full-stack web application that leverages Gemini to simplify the process of itinerary generation. Unlike traditional rule-based or search-based planning systems, Gemini provides contextual reasoning, natural language interaction, and adaptive personalization. The application integrates a Node.js backend with a React (Next.js) frontend, supported by Firebase for authentication and data storage. Through Gemini's generative capabilities, simple user prompts such as destination, duration, budget, and preferences are transformed into structured, day-by-day travel itineraries. External APIs, including Google Maps and Weather APIs, are incorporated to ground the LLM outputs with real-world data, ensuring accuracy and practicality. While the current implementation emphasizes itinerary generation and preference-aware recommendations, this work establishes a foundation for future research in LLM-powered personalization, including predictive trip optimization, collaborative planning via conversational agents, and sustainability-aware travel suggestions. Our The demo of Travel Planner can be viewed in this video: YouTube, Demo Link, Github Repo.

1 Introduction

Modern travel planning is often a complex and time-consuming endeavor. Travelers frequently find themselves switching between multiple platforms to book flights, secure accommodations, explore local attractions, and stay updated on weather conditions. This constant toggling disrupts the planning flow and can make the experience overwhelming. Coordinating group trips adds another layer of complexity, as aligning everyone's preferences, budgets, and schedules can quickly become a logistical challenge[1].

Traditionally, online travel platforms provided static recommendations, leaving users to handle most of the heavy lifting. However, the rise of advanced technologies—particularly large language models (LLMs)—offers the opportunity to create travel solutions that are personalized, adaptive, and dynamic. Rather than treating trip planning as a simple checklist, modern systems can understand user intentions, adjust in real-time, and streamline a process that was once fragmented and tedious.

Recent advances in LLMs have shown remarkable potential for automating aspects of travel planning. Yet, even state-of-the-art models encounter challenges when faced with the multifaceted constraints of real-world trips, which often require balancing hard and soft constraints simultaneously.

Specialized models like POIBERT have achieved notable success in sequential Point of Interest (POI) recommendations by framing them as language tasks, but they primarily suggest paths rather than generating comprehensive, detailed itineraries (Ho and Lim, 2022). Hybrid systems, such as TRIP-PAL, attempt to bridge this gap by using an LLM to interpret user preferences and an automated planner to ensure constraint satisfaction; however, this separation can sometimes lead to a disconnect between the AI's generative capabilities and the final travel plan (de la Rosa et al., 2024).

Our system, Travel Planner, addresses these challenges by consolidating all aspects of trip plan-

[1]*All authors contributed equally to this work.

ning into a single, intelligent platform. At its core, Travel Planner leverages Google's Gemini LLM, a powerful general-purpose language model, as the engine for both reasoning and content generation. This allows the system to produce complete, descriptive itineraries—including tips, recommendations, and packing lists—based on natural language user inputs. The backend, developed using Node.js, orchestrates communication between microservices and external APIs, including real-time weather updates and mapping data. The frontend, built with React and the Next.js framework, delivers a responsive, interactive, and collaborative user experience. This modular, full-stack design not only ensures high performance but also provides scalability, positioning Travel Planner as a practical, end-to-end solution in the evolving landscape of AI-driven travel.

2 System Architecture and Diagrams

The platform's core is its ability to translate simple user inputs into a rich, detailed travel plan. Users begin by specifying their destination, trip duration, budget, and travel style using an intuitive form. The system then processes these preferences to generate a complete travel solution, leveraging advanced reasoning capabilities inspired by recent work on thought decomposition (Wang et al., 2023). Key functionalities include: LLM-powered itinerary generation where the core engine uses Gemini to create contextual plans based on user inputs, incorporating memory mechanisms for personalized recommendations. The platform also features end-to-end booking integration, seamlessly connecting with APIs for arranging flights, trains, and local activities. For example, Integrating Travel Booking Management System Using REST-API demonstrates combining hotel, transport, and ticket booking via a unified API interface (**?**). Another design, Novel Architecture for Distributed Travel Data Integration and Service Provision Using Microservices emphasizes low-latency and scalable API architecture for booking systems (**?**)., emphasizes low latency and scalable API architecture for travel reservation systems.

To ensure relevance, the system provides dynamic itinerary updates by using real-time weather and traffic data to suggest on-the-fly adjustments. Finally, an interactive Gemini LLM chat support system offers real-time assistance, answering user questions and providing instant support through conversational agents powered by large language models. An interactive Gemini LLM chat support system offers real-time assistance, powered by large language models, similar to approaches explored in affective virtual agents (Hasan et al., 2023).

The application is built on a modern, decoupled client-server model designed for scalability and maintainability, consisting of three primary layers: the Frontend, the Backend Server, and External Services. This separation of concerns allows each part of the system to handle its specific tasks efficiently, following best practices in web application architecture.

2.1 Frontend

The frontend is the user-facing part of the application, built with React and the Next.js framework and hosted on Vercel. This stack creates a fast and interactive user interface where travelers input their trip details and view the generated itinerary(Pati and Zaki, 2025). For security and efficiency, the frontend communicates directly with Firebase Auth for user authentication, handling login and sign-up processes without burdening the backend server.

2.2 Backend

The backend server acts as the central processing hub, built as a Node.js API to efficiently handle concurrent requests. Its core components work together to process user input. An API Router serves as the entry point, directing requests to an Input Validator that ensures data integrity. The LLM Processor is the core logic unit, taking validated preferences, formatting them into a detailed prompt using techniques inspired by program-of-thoughts prompting and sending it to the Gemini LLM API. The LLM Processor integrates Gemini into the backend Node.js server (Rajreetesh, 2024). A Database Connector manages all read and write interactions with the Firebase Database, while a Weather Handler communicates with an external Weather API to fetch real-time forecasts.

The functionality of the application is extended by integrating several third-party external services. The Firebase Database, a NoSQL cloud solution, stores persistent data such as user profiles and saved itineraries. The Gemini LLM API provides the advanced LLM capabilities needed for itinerary generation. A dedicated Weather API supplies up-to-date forecasts, and Firebase Authentication offers a secure service for managing user identity.

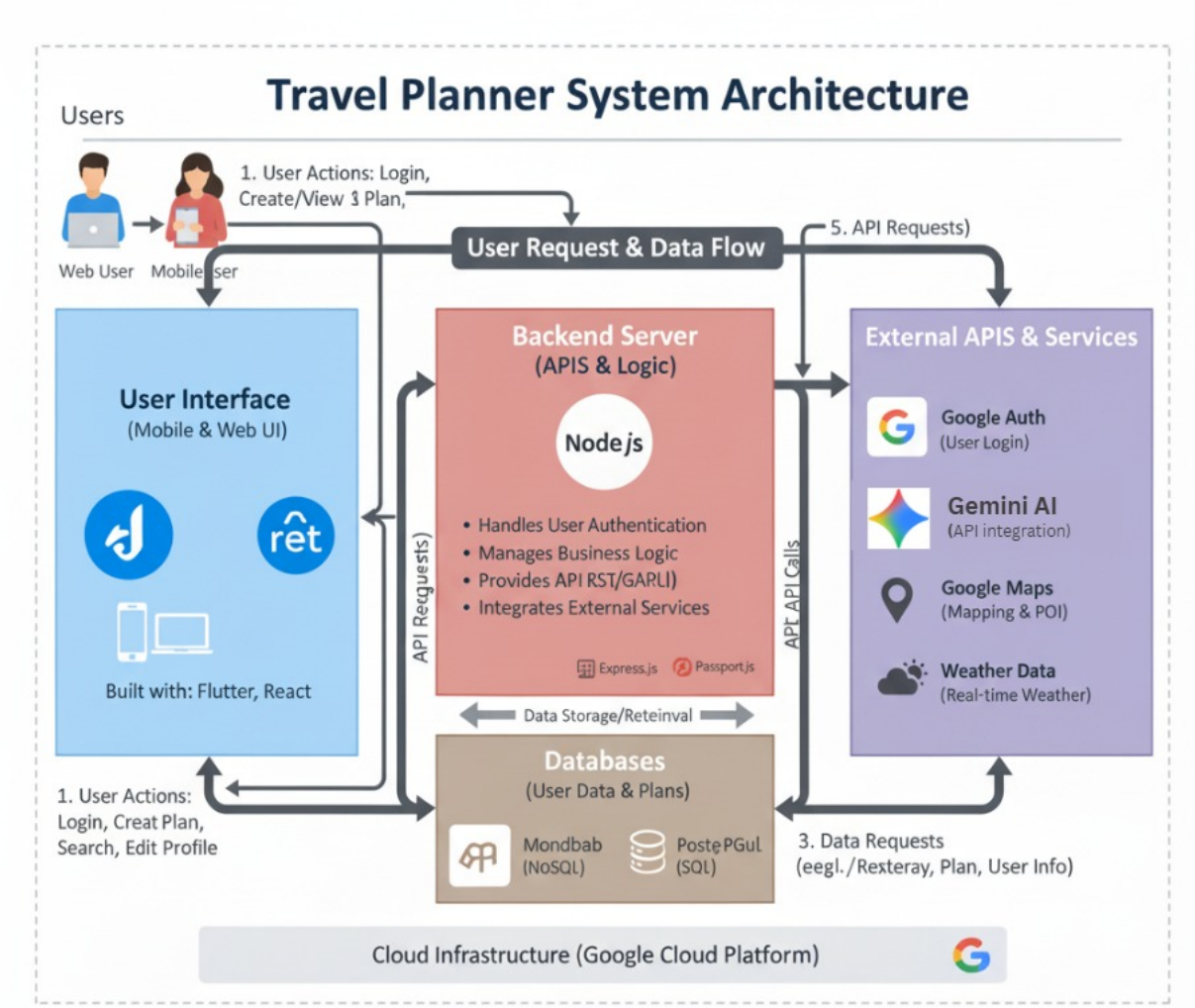

Figure 1: Detailed System Architecture of the Travel Planner, illustrating the data flow between the frontend, Node.js backend, and various external services and APIs.

This architecture enables a seamless end-to-end data flow. A user enters their preferences on the frontend, which sends the data to the backend. The backend validates the input, queries the Gemini and Weather APIs, retrieves any relevant user data from the database, and then combines this information into a comprehensive travel plan. This final plan is sent back to the frontend and displayed to the user.

2.3 Database Schema

The database schema is structured around several key entities to manage application data effectively, drawing inspiration from established tourism recommendation systems. The Users entity stores essential user information, including a unique user ID, name, and email address. The Destinations entity catalogs travel locations with a destination ID, name, and country. Itineraries are stored with a unique ID and are linked to users and destinations via foreign keys; the detailed plan itself is stored in a flexible JSON format. Hotel information, including name, location, price, and rating, is maintained in its own entity. Finally, Bookings are tracked with a booking ID, linking users to specific hotels along with check-in and check-out dates.

3 Testing and Evaluation

We conducted rigorous testing to ensure system robustness and performance, following established evaluation methodologies for tourism systems. Load testing was performed by simulating a high volume of concurrent users with the Firebase Emulator Suite. To optimize performance, the Firestore database was properly indexed and API responses were cached, resulting in significantly improved load times. The LLM (Gemini)-generated outputs were also manually validated for their coherence, accuracy, and relevance to user inputs, similar to

Table 1: Performance Testing Results

Metric	Description	Before	After	Improvement
API Response (ms)	Time for server to respond decreased from 850 ms to 220 ms, making the system much faster.	850	220	74%
DB Query (ms)	Database query time reduced from 420 ms to 95 ms due to indexing, improving data retrieval speed.	420	95	77%
Concurrent Users	The system can now handle 250 users simultaneously, up from 50, showing improved scalability.	50	250	400%
Memory (MB)	Memory usage decreased from 45 MB to 28 MB, making the system more resource-efficient.	45	28	38%

Table 2: Comparison of Travel Planning Features.

Features / Services	Travel Planner	MakeMyTrip	Yatra	EaseMyTrip	Expedia
Smart Itinerary Planning	Personalized itineraries	-	-	-	-
Live Weather	Integrated weather forecasts	Basic	Basic	Basic	Basic
Maps & Navigation	Custom maps	-	-	-	-
Flight & Hotel Booking	Partner APIs	Flights + Hotels	Flights + Hotels	Flights + Hotels	Flights + Hotels
Cost Prediction & Budgeting	budget estimator	Coupons	Coupons	Coupons	Some deals
Offline Support	Itinerary & maps	-	-	-	-
Collaboration / Group Planning	Shared trip planning with friends/family	-	-	-	-

validation approaches used in other LLM-based planning systems. Our evaluation methodology follows best practices in quality assurance for LLM-RAG systems (Ahmed et al., 2025).While most travel websites are built for booking tickets, our Travel Planner is designed for the traveler. The table below shows how our smart, personalized tools help you actually plan your trip, unlike established platforms like MakeMyTrip or Expedia that focus mainly on sales. The system leverages a Large Language Model (LLM) to interpret nuanced user preferences from natural language inputs.

4 Related Work

AI-driven travel planning has progressed from traditional recommendation systems to modern generative agents, with our project positioned at the forefront of this shift. Early work relied on collaborative and content-based filtering to recommend POIs based on popularity or location, but these approaches often ignored individual user preferences. Unlike such systems, our planner employs a generative LLM that produces entirely new itineraries from natural language prompts, including descriptions, tips, and packing lists, rather than merely retrieving items from a fixed database.

Recent models like POIBERT, which frames POI recommendation as a sentence completion task, and TravelPlanner, which benchmarks language agents but shows their struggles with multi-constraint planning, highlight the potential and limitations of sequence modeling approaches. While effective for POI prediction, these models remain specialized. In contrast, our system leverages a general-purpose LLM to generate richer and more adaptable outputs without retraining on tourism data.

Hybrid systems such as TRIP-PAL, which combines LLM-based preference extraction with automated planning for constraint satisfaction, and modular frameworks such as TravelAgent illustrate promising directions. However, our project distinguishes itself by integrating a general-purpose LLM directly into a full-stack architecture (React/Node.js), making it the central reasoning and generation engine. This enables a seamless, interactive, end-to-end travel planning experience, Real-time data with personalized recommendations.

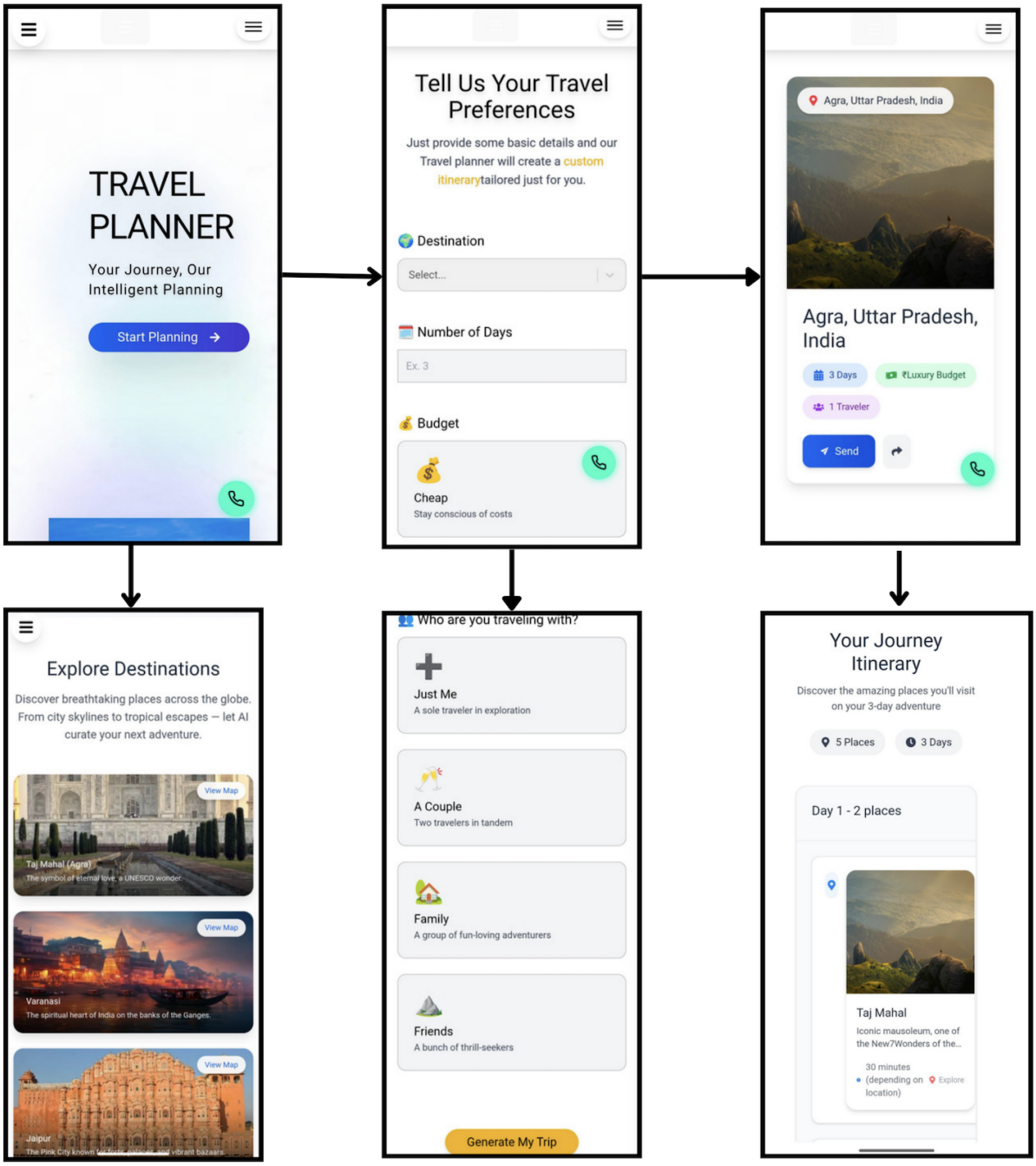

Figure 2: Travel Planner App – User Journey Flow

In summary, our system bridges the gap between research prototypes and practical applications by demonstrating how a general-purpose LLM can drive an end-to-end planning pipeline. This positions the Travel Planner not only as a useful application but also as a research testbed for evaluating LLM-based reasoning in real-world tasks. We see this prototype as an early step toward more intelligent, reliable, and personalized travel planning systems. Future iterations will focus on improving robustness, scalability, and broader user adoption. Moreover, the system highlights important research opportunities in multimodal integration, ethical design, and evaluation of user trust.

5 Future Enhancements and Limitations

Future enhancements to the Travel Planner will focus on intelligence, personalization, and engagement. Predictive analytics will forecast travel trends to suggest optimal booking times, while sustainability features will provide carbon footprint data to promote eco-friendly choices.

For user experience, we plan to integrate AR in the mobile app to overlay historical details on landmarks and translate signs in real time. Social and gamified features will allow users to share itineraries, earn badges, and build a community.

To strengthen adaptability, we will integrate advanced memory architectures and tool-learning capabilities, enabling richer context retention and seamless connections with diverse travel services powered by LLMs.

Limitations

This work presents a prototype system for LLM-driven travel planning, and several limitations remain. At present, the system relies heavily on external APIs (e.g., weather, maps, booking), which creates dependencies on service availability, latency, and cost. Any changes or failures in these services may affect overall reliability.

The system currently uses a general-purpose LLM (Gemini) without domain-specific fine-tuning. While this enables flexibility, it can also result in itineraries that are not fully accurate or that include irrelevant details. In addition, evaluation so far has been limited to a small-scale study and manual inspection; broader user studies are required to validate generalizability.

The current prototype only partially addresses critical issues such as data privacy, hallucinated recommendations, and sustainability concerns.

Since this is an early prototype, we acknowledge that much work remains. We are actively working on advancing the system to make it more robust, accurate, and ethically responsible, with many planned changes and improvements in future iterations.

6 Conclusion

The Travel Planner leverages large language models (LLMs) to simplify the process of travel planning and provide a user-friendly platform. Its modular architecture supports a wide range of features while maintaining flexibility for future extensions, such as predictive analytics, augmented reality, and personalized recommendations.

By integrating advances in LLMs, tool use, and personalization, the system demonstrates how LLM-driven planning can be made practical and accessible for everyday users. As LLM technology continues to advance, systems of this kind have the potential to support travelers with greater confidence, efficiency, and overall satisfaction.

References

Bestoun S. Ahmed, Ludwig Otto Baader, Firas Bayram, Siri Jagstedt, and Peter Magnusson. 2025. Quality assurance for llm-rag systems: Empirical insights from tourism application testing. *Preprint*, arXiv:2502.05782.

Tomas de la Rosa, Sriram Gopalakrishnan, Alberto Pozanco, Zhen Zeng, and Daniel Borrajo. 2024. Trippal: Travel planning with guarantees by combining large language models and automated planners. *Preprint*, arXiv:2406.10196.

Masum Hasan, Cengiz Ozel, Sammy Potter, and Ehsan Hoque. 2023. Sapien: Affective virtual agents powered by large language models. *Preprint*, arXiv:2308.03022.

Ngai Lam Ho and Kwan Hui Lim. 2022. Poibert: A transformer-based model for the tour recommendation problem. *Preprint*, arXiv:2212.13900.

Swostik Pati and Yasir Zaki. 2025. Evaluating the efficacy of next.js: A comparative analysis with react.js on performance, seo, and global network equity. *Preprint*, arXiv:2502.15707.

Rajreetesh. 2024. Integrating google gemini to node.js application. Medium Article. Accessed: Sep. 12, 2025.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *Preprint*, arXiv:2305.04091.