International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

VOYNEX: Intelligent Travel Assistant for Personalized Trip Planning

Yash Patil
Vidyalankar Institute of Technology
Mumbai, India
yn05patil@gmail.com

Niket Lolayekar
Vidyalankar Institute of Technology
Mumbai, India
niketlolayekar10@gmail.com

Niraj Rathod
Vidyalankar Institute of Technology
Mumbai, India
rathodniraj.com@gmail.com

Harshal Dharade
Vidyalankar Institute of Technology
Mumbai, India
harshal.dharade.4.25@gmail.com

Deepali Shrikhande
Vidyalankar Institute of Technology
Mumbai, India
deepali.shrikhande@vit.edu.in

ABSTRACT

Small and medium-sized travel agencies may rely on disconnected tools for itinerary planning, inventory management, booking, and payment processing. This paper presents VOYNEX, a multi-tenant software-as-a-service platform that combines agency resource management, booking workflows, payment verification, and large-language-model-assisted itinerary generation. The application uses a shared PostgreSQL database with organization identifiers for logical tenant separation and supports four roles: traveler, agency staff, agency administrator, and platform administrator. Authentication uses JSON Web Tokens (JWTs) transported in HTTP-only cookies, with role-based authorization and organization-scoped service checks. Itineraries are generated through the Groq API using the Llama 3.1 8B model. Razorpay test-mode payment signatures are verified on the server using HMAC-SHA256. The implementation comprises 15 functional backend modules and 55 API endpoints, with a Next.js frontend, NestJS backend, TypeORM, and PostgreSQL. In functional evaluation, approximately 82% of outputs across 28 itinerary scenarios were judged relevant and practically usable through manual review. No unauthorized access was observed in 45 cross-tenant attempts or 60 role-permission checks. All 30 valid and 10 invalid payment-signature cases produced the expected outcomes. These results demonstrate the feasibility of integrating AI-assisted planning with travel-management workflows. However, generated itineraries are not automatically matched to agency inventory, database-level row security is not implemented, and production-scale performance remains unevaluated.

Keywords: Intelligent tourism, personalized travel planning, multi-tenant SaaS, large language models, role-based access control, travel booking, payment verification.

I. INTRODUCTION

Travel agencies depend on digital systems for itinerary planning, inventory management, booking, customer communication, and payments. Small and medium-sized agencies may perform these activities using separate applications, resulting in fragmented workflows and duplicated information. Research on smart tourism and recommender systems highlights the importance of personalization and decision support in digital travel services [1, 2, 7]. Large language models (LLMs) can interpret natural-language requirements and produce flexible travel plans [3, 8]. However, a plausible generated itinerary is not necessarily bookable: suggested activities, prices, and travel times may not match actual agency inventory or availability.

A further challenge occurs when one application serves several travel organizations. Multi-tenant software-as-a-service (SaaS) systems share infrastructure while logically separating each organization's resources [4, 5]. In a travel platform, trips, bookings, reviews, rentals, products, and payments must be accessed within the correct organizational context. Inconsistent tenant checks can expose customer or business information.

Payment integrity is also essential. A client-side success message must not, by itself, mark a booking as paid. The backend must verify the payment provider's cryptographic signature before changing payment or booking state [12].

This paper presents VOYNEX, a modular monolithic travel-management platform combining AI-assisted itinerary generation, agency-managed resources, organization-scoped data access, role-based authorization, booking operations, and Razorpay test-mode payment verification.

The main contributions are:

1. Tenant-aware architecture employing organization IDs and service layer verification for isolating organizational resources.
2. Itinerary generation process assisted by LLMs, which relies on users' travel preferences.
3. Role-based access controls for travelers, agents, agencies' admin staff, and platform's admin staff.
4. Booking process that includes signature validation via server-side HMAC-SHA256.
5. Functional evaluation of itineraries, tenant isolation, role-based access controls, and signature val-

idation, and an analysis of the system's current limitations.

II. RELATED WORK

A. Intelligent Tourism and Recommendation

Research on smart tourism utilizes information technology in the improvement of service and decision making in the field [1]. The surveys of intelligent tourism recommender systems include the methods of personalized recommendations from structured information, preferences and user interactions [7]. The methods can provide valuable recommendations; however, the quality of recommendations depends on the information and constraints that are used.

The inclusion of natural language interaction by LLMs can help in turning user preferences into a coherent travel itinerary. Transformer architectures are used for the current generation of language models [8], and few-shot learning has already proven the capabilities of large language models in following instructions and examples [3]. However, natural language recommendations may include inaccurate prices, impossible activities, and impractical order of travel.

The recent research includes TravelPlanner for the evaluation of language agents in travel planning, and RETAIL for the investigation of LLM-based travel planning issues [14, 15]. Both papers promote structured output, constraints, and validation of recommendations based on external data. VOYNEX generates natural language itineraries without the validation of every recommendation in terms of agency inventory.

B. Multi-Tenant SaaS and Application Security

Multi-tenancy applications typically make use of different databases, different schemas, or tables with discriminator columns for each tenant [4, 5]. A shared database can help lower the cost of operations, but the borders between tenants should be maintained strictly. The feature called PostgreSQL row-level security (RLS) provides database-wide policies that limit access to rows [6].

Authentication based on JWT tokens and role-based authorization is a standard approach to securing application resources [13]. In addition, payment providers offer signature validation capabilities for payment response and webhook request verification [12]. Such features are used by VOYNEX with JWT authentication, role verification, organization-wide service logic, and server-side payment signature validation.

III. RESEARCH GAP AND PROBLEM FORMULATION

The generation of the itinerary, multi-tenant resource management, and payment verification are usually considered independent technologies. However, in an agency workflow, they should be considered together. An itinerary will help with planning but a booking needs resource verification, its availability, price, permissions and payment status.

The problem solved by this research is the design and evaluation of a multi-tenant travel platform that combines AI-assisted planning, resource management for the organization scope, booking and payment verification.

Platform functionalities are:

1. Create itineraries based on destination, budget, duration, interests, and departure dates provided by the users.
2. Ensure that only those resources which belong to the authenticated organization can be accessed.
3. Apply permissions depending on the user role.
4. Accommodate travel resources under management of agencies.
5. Confirm payment signatures on the server side before registering a transaction.

All the above-mentioned requirements are fulfilled in the current implementation via a combined application flow. Nevertheless, recommendations are not linked with inventory IDs, and therefore, VOYNEX should be treated as a travel management system empowered by artificial intelligence technology.

IV. SYSTEM ARCHITECTURE

The VOYNEX system utilizes a modular monolith architecture – where the functional domains of the application are split into modules and the backend consists of one main application. This architectural pattern enables the separation of business logic without needing to deploy each module as an individual microservice.

A. Technical Stack

For the frontend, we used Next.js 16, React 19, and TypeScript. We utilized Tailwind CSS for styling, Zustand as the state manager for the chosen client-side states, React Query as the data fetching solution, and Axios for HTTP requests. Framer Motion and Recharts provide visual animations and visualization.

As for the backend, NestJS 11 and TypeScript were used. TypeORM is used as an ORM and for the DB connection, and PostgreSQL is used as the persistence layer. Passport.js helps with JWT-based authentication, bcrypt is used for hashing the password, and rate limit control is provided by the NestJS rate limiter package. Llama 3.1 8B makes the API call from the AI part to the Groq API, whereas razorpay makes the payments in the testing environment and Bull from Redis helps us with async tasks.

B. Logical Layers

The app is structured into three logical layers:

- *Presentation layer:* interfaces for travelers, agency employees, agency admin and platform admins.

- *Application layer:* authentication, authorization, trip creation, resource management, booking, payment, reviews and administration.
- *Data layer:* PostgreSQL entities for users, organizations, trips, bookings, payments, reviews, rentals, products, shop orders, notifications, saved plans, wishlists.

C. Backend Modules and Data Model

The backend is implemented using 15 modules and 55 API endpoints for authentication, users, organizations, trips, bookings, payments, reviews, rentals, shop functionality, wishlists, notifications, AI-based trip creation and administration. Data Transfer Objects (DTOs) validation and NestJS's request handling mechanism provide for input validation and exceptions handling. Entities and migration are handled by TypeORM.

The main entities will be Users, Organizations, Trips, Bookings, Payments, Reviews, Rentals, Products, Shop Orders, Notifications, AI-generated Plans, and Wishlist. The organization-specific entities will be linked with tenant IDs. Bookings and payments will be connected to customer and trip-related entities, whereas the generated itineraries will be saved in the `saved_plans` table.

V. AI ITINERARY GENERATION

A. Inputs and Outputs

The inputs for the itinerary generation include a destination, budget (in Indian rupees), trip length, interests, and date of departure. The output from this service should include a trip summary, destination suggestions, daily itinerary, budget breakdown, packing list, and travel tips.

B. Process

In the backend, the travel parameters are entered using a `PlanTripDto` and a prompt is built and passed into the Groq API which returns the answer. A sample prompt is provided below:

```
Make a detailed {duration}-day travel itinerary.
Budget: {budget} INR
Destination: {destination}
Interests: {interests}
Start date: {startDate}

Output:
1. Trip summary
2. Destination recommendations
3. Daily itinerary
4. Budget breakdown
5. Packing list
6. Travel tips

Ensure recommendations are realistic for an Indian
     traveler.
```

The result can be stored in `saved_plans`.

C. Current Constraints

There is no JSON schema in place to check for any structure for model-generated outputs. Additionally, there is no systematic validation of activities, price estimates, or travel-time suggestions against any constraints that exist in the system. There is also no inventory of agencies and their availability being used by the system when generating outputs. This means that outputs can be treated as a way of assisting in planning, and actual booking tasks use application-managed resources. Validation and checking against constraints is planned for later iterations.

VI. MULTI-TENANCY AND SECURITY

A. Authentication and Roles

JWT-based authentication is implemented using Passport.js library. Authentication context includes username, email, role, and organization identifier. The transport mechanism for tokens is HTTP-only cookie. Passwords are hashed using bcrypt hashing algorithm with 10 rounds of salt. JWT strategy verifies tokens sent to the server and checks user's active status on protected routes [13]. HTTP-only cookie mechanism ensures that authentication cookie cannot be accessed by ordinary JavaScript on the client side, but proper configuration is required.

The application provides the following four roles:

- **Traveler:** access to permitted travel discovery, trip planning, booking and customer services.
- **Agency employee:** executes permitted organizational operations.
- **Agency admin:** performs permitted administrative operations on organizational resources.
- **System admin:** performs authorized administrative operations across the entire platform.

Role-based restrictions on the protected endpoints are performed via `@Roles()` decorator and a NestJS role guard depending on the authenticated user's role. Tenant-aware service validation provides an extra level of restriction for tenant-specific services.

B. Tenant Isolation

VOYNEX uses a shared-database approach along with tenant discriminator identifiers. Tenant-specific services use the `organizationId` from the logged-in user's context when accessing tenant-scoped resources. Self-booking restriction configuration also considers the traveler's organization and the organization that owns the resource in question.

Tenant isolation is guaranteed at the application service level. PostgreSQL row-level security feature is not utilized in the current project version, so isolation relies on the consistency of organization checks at service methods and endpoints [4–6]. Role-based authorization determines which operations a user may perform; tenant isolation determines which organization's resources the user may access.

C. Rate Limiting

The configured rate limit thresholds are 10 requests per minute globally per IP address, 5 requests per minute

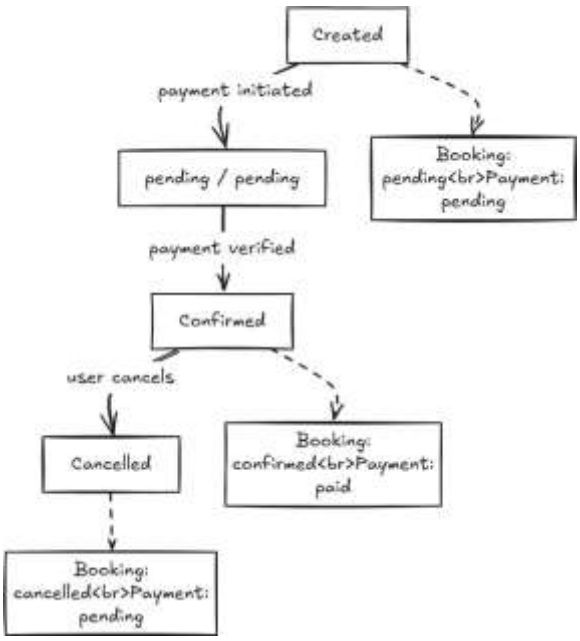


Figure 1: Booking state transitions in VOYNEX.

for login pages, and 30 requests per minute for administrative pages. These rate limit thresholds help prevent too many requests being sent but do not replace any other security measures.

VII. BOOKING AND PAYMENT WORKFLOW

A. Booking Validation

Prior to booking creation, validation of the selected trip, its active status, travel date, organization context, and the amount is conducted by the backend. Additionally, the backend validates the travel organization compared to the owner of the resource. A validated booking leads to a confirmation email and an in-app notification.

B. Payment Order and Signature Validation

The payment is integrated with Razorpay in testing mode. The backend creates an order based on the booking amount in Indian Rupees converted into paise and creates a payment record with the ”pending” status. The frontend launches the Razorpay Checkout and sends the order ID, payment ID, and the signature to the backend after checkout.

The backend validates the signature using HMAC-SHA256 algorithm and the secret key, which is stored on the server and should never be visible for the frontend [12]. If the signature is valid, it allows updating the payment state and the related booking state. The invalid signature does not allow making a successful payment transition.

C. Booking State Transitions

Fig. 1 shows the booking state transitions in VOYNEX that depend on the results of payment verification.

D. Webhook handling

Webhooks validation involves validating the incoming `x-razorpay-signature`. Webhook handling in events is not implemented in the current version, so the asynchronous payment-state recovery is not fully implemented. In production implementation, webhooks handling, idempotency, reconciliation, and recovery from interruptions of the client session must be implemented.

E. Workflow Summary

Desired workflow:

1. The client selects an agency managed travel resource and sends a booking request.
2. The backend checks the validity of the resource, organization, travel date and the amount.
3. The payment order is created and the payment is set to **pending**.
4. The client pays using Razorpay.
5. The backend verifies the signature and updates payment and booking states.

VIII. EXPERIMENTAL METHODOLOGY

The evaluation focuses on functional correctness and integration of itinerary generation, tenant isolation, role-based access control, and payment verification. It is a practical assessment of the implemented application rather than a comprehensive production-scale benchmark.

A. Experimental Setup

Testing took place on a 64-bit Windows 11 machine with 16 GB RAM, Intel Core i5-12450H processor, and 512 GB NVMe SSD. Software environment consisted of Node.js version 26.8.1, PostgreSQL version 18.6, and Redis version 8.10.1. Payment testing was done using Razorpay test environment without any monetary transactions.

B. Evaluation Procedure

Itinerary planning was tested under scenarios differing in destination, budget, duration, interests, and start date. Generated itineraries were checked manually for relevance, structure, and practicality. Testing of tenant isolation was based on requests dealing with multiple tenants, users, and resources connected to tenants. Testing of role-based authorization included checking permissions and restrictions associated with all four roles. Payment testing included creating orders, signing with valid and invalid signature, and payment states.

While testing the behavior of the API has been done, no load testing campaign with measured latency distribution, throughput, concurrency, and error rates has been performed.

IX. RESULTS

These results are concerned with the observed behavior during development and testing phase and must not be regarded as a security certification or demonstration of production capability.

A. Itinerary Generation

For 28 trip conditions, the AI service produced summaries, planned daily activities, budget estimations, packing advice, and travel tips. The average generation time was around 1.8 to 2.4 seconds and 82% of the output was considered relevant and practical by manual evaluation.

While the above result implies that prompt-based generation can give initial plans of practical use, it must be noted that this is an estimate made manually and not a confirmed measure of accuracy. Prices, travel duration, activity availability, and destinations were not checked against the agency's catalog.

B. Tenant Isolation and Role Permissions

In 45 cases of cross-tenant access checks and 60 checks of role permissions, there were no instances of unauthorized access being made. Thus, it can be concluded that service operations under test adhered to the intended organizational and role limitations in the tested scenarios. The tests above do not prove the absence of authorization vulnerabilities for each endpoint, request sequence or concurrency scenario. The tests carried out were functional rather than a full security audit. The database-level RLS was not enabled during testing.

C. Payment Verification

There were 40 cases of payment signature verification, 30 of which involved the valid signature while 10 were invalid. In all cases, valid signatures have been accepted and invalid rejected. The results above confirm the proper functioning of the signature verification process. Nevertheless, there are some problems with event-specific webhook handling, which limits the possibility to recover from asynchronicity in payment flow.

D. Performance

Several API operations were checked via implementation of authentication, validation, tenant and role control. Distributions of numerical latency, number of requests per second, sustained concurrency capabilities and API errors were not evaluated.

X. DISCUSSION

The evaluation proves the possibility of implementing the functionality of AI-assisted planning, multi-tenant resource management, role-based authorization, and payment verification in one modular system.

The results of the itinerary evaluation prove that the natural language generation can be used for creating the initial travel plans by preference. However, even though the language generated is plausibly correct, there is no proof of its factual accuracy and bookability. Prices offered by AI algorithms might be estimations and proposed activities might not exist, thus, the itinerary created should be separated from booking records until proven.

Table 1: Qualitative Functional Scope Comparison

Capability	VOYNEX
Itinerary creation via AI	Yes
Multi-tenant organizations model	Yes
Agency resources management	Yes
Booking and payments process	Yes
Access based on roles	Yes
Verified purchase associations	Yes
Auto-matching of inventory with AI	Not yet
Row security in database	Not yet
Full payment webhook support	Not yet

The results of the tenant and roles evaluation prove that there were no unauthorized access attempts within the tested scenarios. Thus, it might be possible to use authentication of organizations and checks at the service layer. However, the approach relies on consistency of this process within all tenant-scope operations. PostgreSQL RLS might be used as an additional enforcement mechanism [5, 6].

Signature verification on the server side prevents the tested transaction from processing invalid signatures. Production deployment would entail webhook event handling verification, idempotent changes, reconciliation, and handling interruption during sessions. Booking and payment state change design should aim to prevent inconsistent states in case of failures between such operations.

A modular monolith is useful in keeping functional components well-organized without the need for additional costs associated with microservices deployment. Separation into services in the future should be triggered by actual need and not assumed advantages [9–11].

XI. COMPARISON WITH EXISTING APPROACHES

Table 1 provides a qualitative comparison of the scope of functions of VOYNEX and selected travel applications. This is an opinionated assessment and not a formal benchmark or audit of the current capabilities of the respective products.

VOYNEX is an integration of AI-enabled trip planning and resources management for agencies, organization administration, bookings, and payments. Consumer travel applications can specialize in personal trip organization and marketplace operations whereas SaaS frameworks are domain-neutral.

XII. LIMITATIONS AND FUTURE WORK

However, the current system has a number of deficiencies. First, the LLM outputs natural language text instead of a formally validated structured docu-

ment, meaning that completeness and structure of the answer cannot be guaranteed. Second, there is no automatic matching of the recommendations against agency inventories, availability, prices, and resource IDs. Third, isolation between tenants happens at the level of service, since PostgreSQL RLS is not implemented. Fourth, webhook signatures are verified, but the event-based payments are not fully supported. Finally, the security audit, a large-scale benchmark of the AI, and performance measurements are not performed.

Further research will tackle these issues through the following steps:

- JSON output, schema validation, and explicit checking of the constraints for itineraries;
- matching of the recommendations against agency inventories, availability, prices, and resource IDs;
- webhook processing, idempotence, reconciliation, and recovery from an interrupted payment session;
- PostgreSQL RLS and adversarial testing of tenant-scoped endpoints;
- larger itineraries datasets, human review criteria, repetitive latency measurements, controlled load test, API errors rate reporting.

If workload characteristics call for it, selected domains can be separated from the modular monolith to standalone services in the future.

XIII. CONCLUSION

This paper described VOYNEX, a multi-tenant travel management system featuring AI-assisted itinerary planning, resource management by agencies, booking, and payment signature validation in a modular monolithic structure. The implementation makes use of Next.js, NestJS, TypeORM, PostgreSQL, the Groq API, and Llama 3.1 8B along with Razorpay test-mode payments. There are four roles available in the application, with organization-specific service checks being performed to achieve logical tenant isolation.

Functional testing revealed that about 82% of outputs for 28 scenarios of itinerary generation were considered relevant and usable after manual inspection. No unauthorized access was detected during 45 attempts at cross-tenant access and 60 tests involving role permissions. For all 30 valid and 10 invalid payment signature inputs, the appropriate responses were received. This suggests the possibility of integrating an AI-assisted solution into a traditional travel management system. However, automated inventory matching, row-level security in the database, full processing of webhooks, and performance testing are still left undone.

REFERENCES

[1] U. Gretzel, M. Sigala, Z. Xiang, and C. Koo, "Smart tourism: foundations and developments," *Electronic Markets*, vol. 25, no. 3, pp. 179–188, 2015.

[2] D. Buhalis and R. Law, "Progress in information technology and tourism management: 20 years on and 10 years after the Internet—the state of eTourism research," *Tourism Management*, vol. 29, no. 4, pp. 609–623, 2008.

[3] T. B. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[4] J. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: maintenance dream or nightmare?" in *Proc. 4th Int. Joint ERCIM/IWPSE Symp. Software Evolution*, pp. 88–92, 2010.

[5] Microsoft, "Architect multitenant solutions on Azure: tenancy models overview." [Online]. Available: `https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/approaches/overview`

[6] PostgreSQL Global Development Group, "Row security policies." [Online]. Available: `https://www.postgresql.org/docs/current/ddl-rowsecurity.html`

[7] J. Borra`s, A. Moreno, and A. Valls, "Intelligent tourism recommender systems: A survey," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7370–7389, 2014.

[8] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017.

[9] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[10] S. Newman, *Building Microservices*, 2nd ed. O'Reilly Media, 2021.

[11] S. Newman, *Monolith to Microservices*. O'Reilly Media, 2019.

[12] Razorpay, "Razorpay API documentation—payments and webhooks." [Online]. Available: `https://razorpay.com/docs/api/`

[13] OWASP Foundation, "JSON Web Token for Java Cheat Sheet." [Online]. Available: `https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html`

[14] J. Xie, K. Zhang, J. Chen, T. Zhu, R. Lou, Y. Tian, Y. Xiao, and Y. Su, "TravelPlanner: A benchmark for real-world planning with language agents," in *Proc. 41st Int. Conf. Machine Learning*, PMLR, vol. 235, pp. 54590–54613, 2024.

[15] B. Deng *et al.*, "RETAIL: Towards real-world travel planning for large language models," in *Proc. 2025 Conf. Empirical Methods in Natural Language Processing*, pp. 14870–14902, 2025.