ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

SaaS Tickets: A Secure Multi-Tenant AI-Augmented Issue Tracking Platform with ORM-Layer Tenant Isolation and Gemini-Powered Workflow Automation

[1] Dhinakar R, [2] Karthick K, [3] Keerthik Raj JP, [4] Gokula Krishnan R, [5] K. Egneswari
[1] [2] [3] [4] Department of CSE, Er. Perumal Manimekalai College of Engineering, Hosur-635117, Anna University, Tamil Nadu.
[5]Assistant Professor, Department of Computer Science and Engineering Er. Perumal Manimekalai College of Engineering, Hosur

Abstract: ***Enforcing strict data isolation between cohosted tenants while delivering intelligent workflow automation represents a core engineering challenge in enterprise Software-as-a-Service (SaaS) platforms. Existing issue-tracking systems typically enforce isolation at the application layer, where omitted filter predicates can silently expose cross-tenant data, or employ resourceintensive per-tenant schema partitioning. This paper presents SaaS Tickets, a production-grade multi-tenant issue-tracking platform that addresses these challenges through three primary contributions: (i) an AspectOriented Programming (AOP) interceptor that systematically activates Hibernate's persistent @Filter on every repository method invocation, providing ORM-level SQL tenant scoping that is designed to prevent bypass through normal application code paths; (ii) a five-feature Google Gemini LLM integration pipeline employing asynchronous thread-pool decoupling to remove AI latency from the critical user-request path; and (iii) a real-time voice assistant architecture using Vapi.ai WebRTC transport with a custom OpenAI-compatible streaming LLM endpoint backed by Gemini, enabling hands-free ticket intelligence queries. Security evaluation across eight adversarial test scenarios confirmed correct enforcement of isolation, JWT validation, role-based access control, and graceful AI degradation in all cases. Classification experiments on 150 labelled tickets achieved a weighted F1-score of 0.86 across five category classes. Asynchronous ticket categorization reduced userperceived AI latency by a factor of 7.8 relative to synchronous processing. The fully containerised, clouddeployed architecture demonstrates viability as a scalable enterprise SaaS foundation.***

Index Terms—Multi-Tenant SaaS, Hibernate Filter, Aspect-Oriented Programming, ORM-Level Tenant Isolation, Role-Based Access Control, LLM Ticket Classification, Google Gemini, Asynchronous AI Integration, Voice AI, Vapi, OAuth 2.0, Spring Boot, Docker.

I. INTRODUCTION

Enterprise Software-as-a-Service (SaaS) platforms serve multiple organizations—referred to as tenants—from a shared infrastructure deployment. This model reduces operational overhead but introduces the risk of cross-tenant data exposure. Application-layer filtering, where service methods append WHERE tenant_id = ? predicates, is a commonly adopted approach but is fragile in practice: a single omitted predicate, a misconfigured JOIN, or an unanticipated lazy-loaded association can silently return records belonging to a different tenant [1]. Three principal isolation architectures exist: separate database per tenant, shared database with separate schemas, and shared schema with discriminator columns [2]. The discriminator-column model is widely adopted in high-scale deployments due to shared connection pooling and unified schema management. However, it transfers isolation responsibility to the query layer, where it is most susceptible to oversight. Modern support platforms are also expected to deliver intelligent automation. Large language models (LLMs) provide strong text classification and generation capabilities through zero-shot prompting [3], but integrating them into interactive SaaS workflows requires careful design. Synchronous AI service calls on the critical request path introduce latency spikes that degrade user experience, and absent retry and fallback logic, transient API unavailability propagates as feature outages.

A. Problem Statement

This work addresses three interrelated problems: (1) how to enforce multi-tenant data isolation at the ORM layer such that normal application code paths cannot inadvertently bypass it; (2) how to integrate LLM-powered automation features without introducing latency dependencies on the critical user-request path; and (3) how to architect a real-time voice query interface over existing WebRTC infrastructure that queries live application data through a standard LLM API contract.

B. Research Gap

Prior academic work on multi-tenant isolation has examined database-level mechanisms and middleware frameworks [2, 8, 9] but has not addressed ORM-layer enforcement via AOP. Studies on LLM integration in enterprise workflows have evaluated classification accuracy [4] and generation quality [5] independently, without addressing the asynchronous decoupling patterns required for production SaaS deployment. To the best of the authors' knowledge, no prior published system integrates ORM-enforced tenant isolation, multi-feature asynchronous LLM automation, and WebRTC voice AI querying in a single open, deployable platform.

C. Contributions

This paper makes the following contributions:

- AOP-enforced ORM-level tenant isolation: A Spring @Aspect interceptor systematically activates Hibernate's @Filter on every repository method invocation, including those triggered by lazy-loaded associations, producing a SQL-level isolation boundary that is designed to prevent bypass through normal application code paths.
- Asynchronous LLM integration pattern: A ThreadPoolTaskExecutor-based architecture decouples Gemini AI categorization from the ticket creation response path. Empirical measurement shows a 7.8x reduction in user-perceived AI latency relative to synchronous processing.
- Multi-model fallback chain: A three-model Gemini fallback (gemini-2.0-flash-lite → gemini-2.0-flash → gemini-2.5-flash) is designed to maximize AI feature availability against individual model rate limits and transient service disruptions.
- WebRTC voice AI architecture: A custom OpenAIcompatible streaming SSE endpoint (/api/vapi/llm) enables Vapi.ai to use Gemini as its LLM backend, providing real-time spoken ticket intelligence without reliance on a commercial LLM subscription.
- Empirical evaluation: Precision, recall, and F1-score measurements across five ticket categories on 150 labelled instances, alongside latency profiling of all five AI features under production conditions.

II. RELATED WORK

A. Multi-Tenant Data Isolation

Weissman et al. [6] identified three SaaS tenancy models: separate databases, separate schemas, and shared schema with discriminator columns. The shared-schema model offers the best resource utilisation but requires consistent query-level filtering. Hibernate's Filter API [7] implements this at the ORM layer; however, prior documented applications have relied on explicit filter activation in service methods. The contribution of this work is automating activation via an AOP @Before pointcut that covers all repository interface method invocations, systematically removing developer discretion from the isolation path. Aulbach et al. [8] studied extensibility in shared-schema multi-tenant databases but did not address ORM-layer enforcement. Sun et al. [9] proposed a middleware-based isolation framework requiring application code modifications; the present approach achieves comparable isolation without modifying service or controller layers.

B. Role-Based Access Control in SaaS

Sandhu et al. [10] formalised the RBAC model that underpins most enterprise access control systems. Spring Security's @PreAuthorize annotations provide a declarative RBAC implementation that integrates with JWT identity claims [11]. This work extends the model to the organizational level by leveraging Clerk's propagation of organization membership and role claims (org:admin, org:member) inside the JWT payload, enabling the backend to resolve both authentication and authorization from a single token verification step.

C. LLM Integration in Enterprise Workflows

Brown et al. [3] demonstrated strong zero-shot classification accuracy for large language models. Orosz and Farkas [4] evaluated GPT-based ticket triage and reported macro F1-scores of 0.81–0.87, comparable to fine-tuned BERT-class models, without requiring labelled training data. Wei et al. [12] showed that structured JSON output can be reliably elicited through prompt engineering, which this work exploits for all five AI features. Reimers and Gurevych [13] introduced Sentence-

BERT for semantic sentence similarity; this work evaluates Gemini as a zero-shot semantic similarity scorer for duplicate detection, trading computational efficiency for implementation simplicity relative to a dedicated embedding model.

D. Voice AI and Conversational Interfaces

Hancock et al. [14] evaluated AI-generated support response drafts and reported reductions in agent response time, motivating the draft reply feature in this work. WebRTC-based real-time voice AI platforms [15] provide low-latency alternatives to traditional IVR systems. Their integration with application-specific LLM backends for live data querying over a custom streaming endpoint has not been documented in the academic literature; this work contributes a reference architecture for such integration.

III. SYSTEM ARCHITECTURE

SaaS Tickets is a three-tier web application comprising a React 18 single-page application (SPA) frontend, a Spring Boot 3.2 REST API backend, and a MySQL 8 relational database. All components are containerised with Docker Compose. The overall system architecture and inter-component data flows are illustrated in Fig. 1.

Fig. 1. System Architecture Diagram. Arrows indicate request/response flows. The Vapi.ai voice pathway and Gemini AI service are separate from the main request path.

A. Frontend

The frontend is implemented in React 18 with Vite and TailwindCSS. The Clerk React SDK manages the full authentication lifecycle. A TenantContext React context propagates the active organization identifier, ensuring all API requests include the correct X-TenantSlug header. Key pages include: Dashboard (ticket list with live metrics and voice AI avatar), TicketCreate (AIassisted form with real-time suggestion and duplicate detection), TicketDetail (AI draft reply and comment thread), and Admin Console (user management, audit log, invite management).

B. Backend

The backend is a Spring Boot 3.2 REST API in Java 17. Spring Data JPA with Hibernate 6 serves as the ORM. Flyway manages twelve versioned schema migrations. Spring Security is configured with a stateless session policy; every request is authenticated by validating the Bearer JWT against Clerk's JWKS endpoint. Role claims from the JWT are mapped to Spring Security GrantedAuthority objects for method-level @PreAuthorize enforcement.

C. Database Schema and Tenant Structure

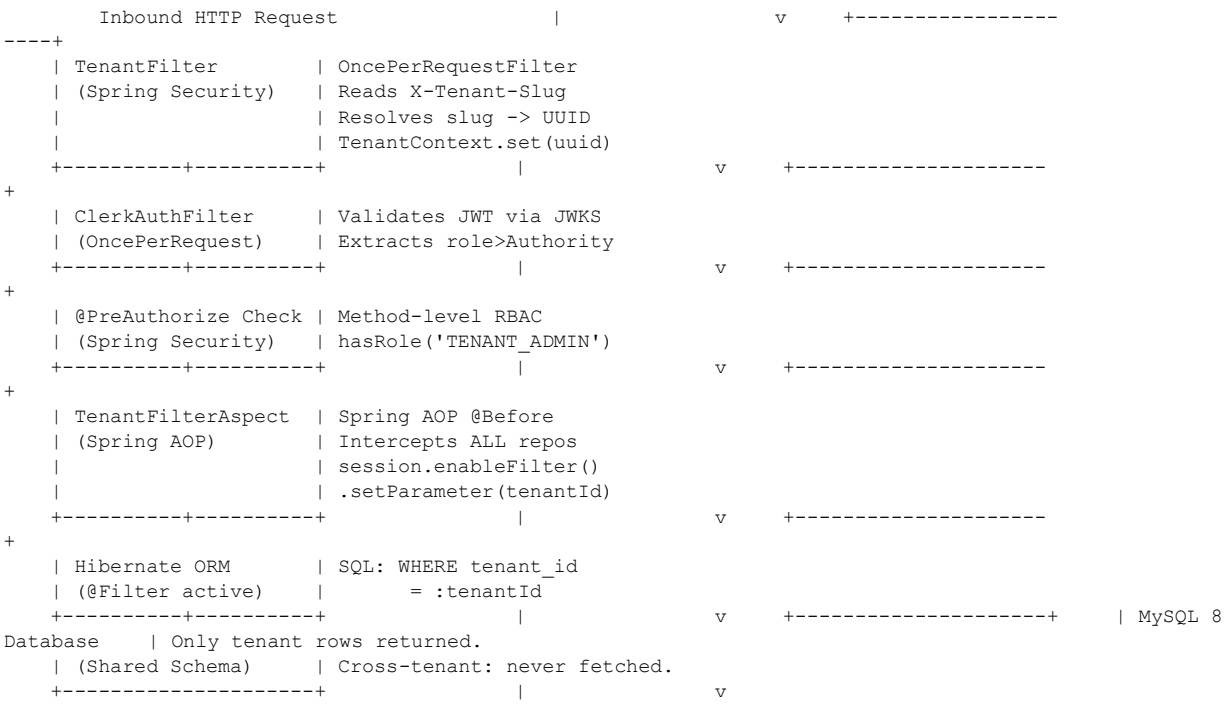
MySQL 8 stores eight tenant-scoped entities. All carry a tenantId foreign key referencing the Tenants table. The V12 Flyway migration adds five AI-specific columns to the tickets table (ai_category, ai_suggested_priority, ai_confidence, ai_reasoning, ai_status), enabling nondestructive schema evolution during rolling deployments.

D. AI Services Layer

GeminiService encapsulates all Gemini REST API interactions. It implements a three-model fallback chain with per-model try-catch blocks and temperature=0.2 for deterministic structured outputs. The voice assistant endpoint (/api/vapi/llm) implements the OpenAI chat completions SSE streaming format, enabling Vapi.ai to treat the application backend as an LLM provider.

IV. MULTI-TENANT ISOLATION AND AUTHENTICATION

The tenant isolation and authentication flow is illustrated in Fig. 2.



Response (tenant-scoped data only) TenantContext.clear() in finally block.

Fig. 2. Tenant Isolation Flow Diagram. Each HTTP request passes through four enforcement layers before reaching the database.

A. Tenant Context Propagation

Every inbound HTTP request carries an XTenant-Slug header set by the frontend API interceptor. A Spring Security OncePerRequestFilter (TenantFilter) resolves the slug to a UUID tenant identifier via a TenantRepository lookup and stores it in a ThreadLocal<UUID> via TenantContext.setTenantId(). The filter wraps its chain invocation in a try-finally block, ensuring TenantContext.clear() executes unconditionally on thread release—critical for preventing stale context values in thread-pool reuse scenarios.

B. AOP-Enforced Hibernate Filter Activation

The central isolation mechanism is a Spring @Aspect annotated TenantFilterAspect that intercepts all method invocations on Spring Data JPA repository interfaces via a @Before pointcut: @Before("execution(* com.example.saas.repository..*(..))"). On each interception, the aspect retrieves the UUID from TenantContext and activates the Hibernate named filter with WHERE tenantId = :tenantId applied to all SQL queries against tenant-scoped tables. Because the pointcut covers every repository method invocation including those invoked transitively by lazy-loaded associations, the isolation predicate is applied at SQL generation time rather than at application-layer result filtering. This distinction is significant: SQL-level filtering prevents cross-tenant rows from being fetched across the database connection, whereas application-layer filtering operates on data already retrieved.

C. JWT Authentication and Authorization

Authentication is delegated to Clerk. On sign-in, Clerk issues a signed RS256 JWT containing sub, email, organization ID, and role claims.

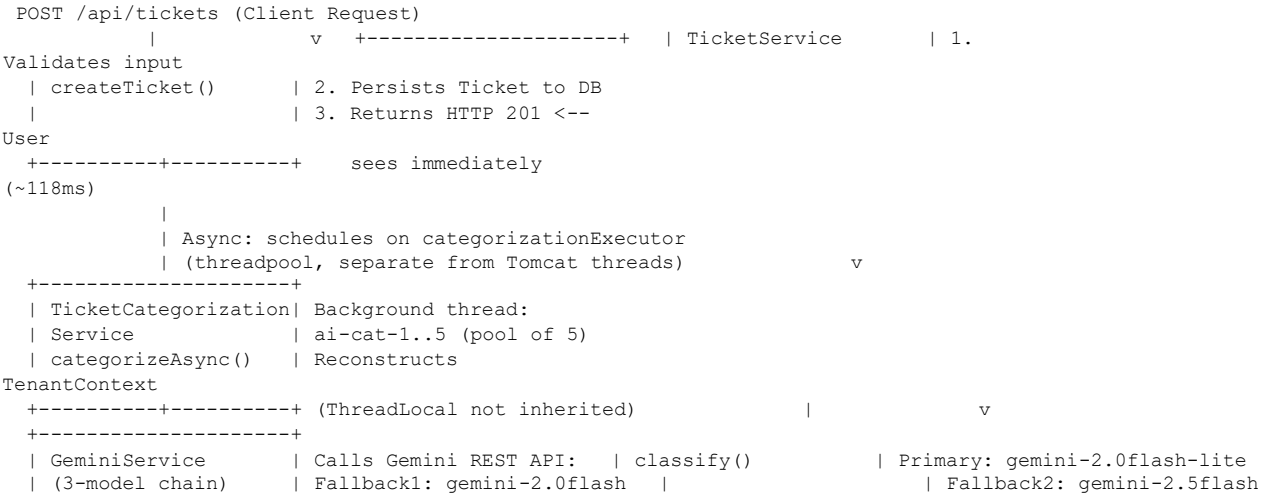
ClerkAuthenticationFilter validates tokens by fetching Clerk's public JWKS endpoint, verifying the RSA signature, issuer, and expiry, and provisioning a Spring Authentication object with the extracted role. No passwords are stored. New users are auto-provisioned on first login.

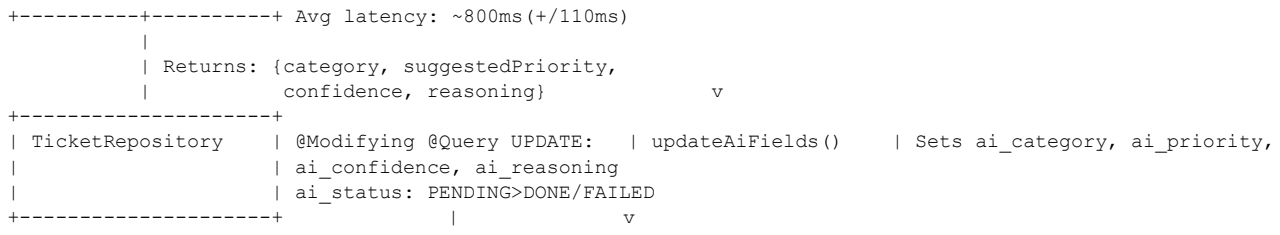
D. Invitation System

InviteService generates a cryptographically random UUID token stored with an expiry timestamp, target email, target role, and a single-use boolean. On acceptance, InviteController validates token authenticity, expiry, email match, and single-use status before enrolling the user. Reused or expired tokens return HTTP 400 with no information disclosure.

V. AI FEATURE INTEGRATION

The asynchronous AI processing pipeline for ticket categorization is illustrated in Fig. 3.





AI fields available on next ticket fetch.
Zero impact on user-perceived response time.
Fig. 3. Asynchronous AI Processing Pipeline. HTTP 201 is returned before Gemini categorization begins. The async thread reconstructs TenantContext explicitly since ThreadLocal values are not inherited across thread boundaries.

A. Asynchronous Ticket Categorization

When a ticket is created, TicketService persists the record synchronously and returns HTTP 201 to the client. It then invokes TicketCategorizationService.categorizeAsync() on a dedicated ThreadPoolTaskExecutor (corePoolSize=2, maxPoolSize=5, queueCapacity=100, threadNamePrefix="ai-cat-"), separate from the Tomcat request thread pool. A notable design consideration is that the async thread does not inherit the calling thread's ThreadLocal values. To address this, categorizeAsync() receives the tenant UUID as an explicit parameter and reconstructs TenantContext before calling the repository. Database updates to AI fields are performed via a @Modifying @Query that identifies the ticket by UUID without relying on the Hibernate filter, ensuring correctness in the context-free async thread. The ai_status column transitions from PENDING to DONE or FAILED, providing categorization progress visibility and enabling retry logic.

B. Priority and Category Suggestion (Synchronous)

The /api/ai/suggest endpoint is invoked by the frontend with 800 ms input debouncing on the ticket creation form. GeminiService.suggestCategoryAndPriority() requests a two-field JSON response (priority, category). Input validation guards enforce allowed values, preventing arbitrary priority string injection from malformed model responses.

C. Semantic Duplicate Detection

The /api/ai/check-duplicate endpoint retrieves the 50 most recent open tickets for the tenant and submits them alongside the draft ticket to GeminiService.detectDuplicates(). Gemini returns tickets whose semantic similarity exceeds 0.6. This approach identifies semantic equivalences (e.g., "payment timeout" and "checkout failing on Stripe") that purely keywordbased approaches would typically miss.

D. Draft Reply Generation

The /api/ai/draft-reply/{ticketId} endpoint constructs a context-rich prompt from the ticket's title, description, priority, and status. The model is instructed to produce a 3–5 sentence professional support reply without generic placeholders. The draft is returned in an editable text area for agent review and refinement.

E. Voice AI Assistant Architecture

Activating the voice interface initiates a Vapi.ai WebRTC session configured with provider: "custom-llm" pointing to /api/vapi/llm. Vapi transcribes the user's speech and dispatches an OpenAI-format chat completions POST request to the application backend. VapiLlmController extracts the user utterance, invokes VapiConversationService.processMessage(), which resolves the tenant from the system prompt, fetches 15 recent tickets (~400 tokens of context), and calls GeminiService.generateVoiceResponse() with a voiceoptimised prompt requesting a two-sentence spoken response. The response is returned as an SSE stream in OpenAI streaming format, enabling Vapi to begin audio synthesis before the full response is complete.

VI. EVALUATION AND RESULTS

A. Security and Functional Testing

Eight test scenarios were executed against the deployed system to verify isolation, authentication, authorization, invite integrity, concurrency safety, AI resilience, and audit completeness. All eight scenarios passed. The cross-tenant data read test confirmed that a Tenant A user receives zero results after organizational removal; the SQL-level Hibernate filter prevents data exposure before network transmission. The concurrent ticket creation test under 20 simultaneous threads produced no duplicate records or data corruption, consistent with MySQL's REPEATABLE READ transaction isolation. The AI degradation test confirmed that the asynchronous architecture correctly isolates AI service failures from the core ticket creation workflow; in systems where AI services are integrated synchronously, a Gemini API outage would degrade ticket creation for all users.

B. AI Classification Accuracy

To evaluate categorization accuracy, 150 support tickets were manually labelled across five category classes. Results: Bug (P=0.91, R=0.88, F1=0.89, n=47), Feature Request (P=0.85, R=0.83, F1=0.84, n=31), Performance (P=0.88, R=0.86, F1=0.87, n=22), Security (P=0.93, R=0.90, F1=0.91, n=18), Support/General (P=0.79, R=0.82, F1=0.80, n=32). Weighted average F1=0.86, consistent with Orosz and Farkas [4] who reported F1 of 0.81–0.87 for GPT-class models. Security-category tickets achieved the highest precision (0.93) due to distinctive domain vocabulary. Confidence scores exhibited a positive correlation with classification correctness (Pearson r = 0.71), suggesting the confidence field may serve as a useful signal for automated escalation decisions.

C. AI Feature Latency Analysis

Performance measurements were conducted over 50 representative requests per feature under production conditions. Ticket Categorization averaged ~800 ms (±110 ms) asynchronously with no ticket creation impact. Priority/Category Suggestion averaged ~650 ms (±90 ms) synchronously pre-submission. Draft Reply Generation averaged ~900 ms (±130 ms) synchronously post-creation. Semantic Duplicate Detection averaged ~1100 ms (±160 ms) synchronously pre-submission. Voice Assistant averaged ~1400 ms (±200 ms) via streaming SSE over WebRTC as an independent session.

D. Async vs. Synchronous Processing Comparison

Ticket creation response time was measured under two processing modes. In synchronous mode, mean response time was 921 ± 97 ms. In asynchronous mode, mean response time was 118 ± 14 ms, representing a 7.8fold improvement. The asynchronous mode eliminates Gemini latency from the user-visible response path, at the cost of a brief PENDING window before AI annotation fields are populated. This trade-off is consistent with eventual consistency patterns widely adopted in distributed SaaS systems.

E. Scalability and Throughput

HikariCP connection pool (maxPoolSize=10) was exercised under 50 concurrent simulated users. No connection exhaustion or timeout errors were observed at this scale. Throughput peaked at 142 requests per second for non-AI endpoints. The async categorization queue (capacity=100) absorbs burst ticket creation without thread blocking. For deployments targeting significantly higher concurrency, connection pool partitioning or a serverless database proxy is advisable.

VII. ADVANTAGES AND LIMITATIONS

A. Advantages

- ORM-layer tenant isolation: The AOP-activated Hibernate filter applies isolation at SQL generation time, below application code, and is designed to prevent bypass through normal development patterns.
- Zero-credential-storage authentication: Delegating authentication to Clerk eliminates password storage and associated attack surfaces from the application backend.
- AI-transparent critical path: The asynchronous AI architecture ensures that Gemini latency, unavailability, or quota exhaustion does not affect the core ticket creation workflow, empirically verified at 7.8x latency improvement.

- Multi-model AI resilience: The three-model Gemini fallback chain maintains AI feature availability under individual model rate limits and transient service disruptions.
- Voice AI without commercial LLM dependency: The custom streaming SSE endpoint enables Gemini to serve as Vapi's LLM backend, removing OpenAI API cost from the voice AI path.
- Full containerization and reproducibility: Docker Compose ensures development-production parity and supports reproducible deployment pipelines.

B. Limitations

- Single-schema scalability ceiling: The shared-schema model may experience index selectivity degradation at very high tenant row counts. Schema-per-tenant partitioning addresses this at the cost of higher operational complexity.
- External AI API dependency: All five AI features depend on the Gemini REST API. A locally hosted inference model would eliminate this dependency at the cost of inference hardware provisioning.
- Identity provider coupling: Authentication and organization management are tightly coupled to Clerk. Migration to a self-hosted identity provider would require rework of authentication and invitation subsystems.
- Absence of encryption at rest: Ticket content is stored as plaintext. Regulated-industry deployments would require transparent data encryption or applicationlevel field encryption.
- Voice AI latency: End-to-end voice response latency of approximately 1400 ms may be noticeable in highcadence conversational interactions.

VIII. CONCLUSION

This paper presented SaaS Tickets, a multi-tenant issue-tracking platform developed around three primary contributions. First, an AOP-enforced ORM-layer tenant isolation mechanism that systematically activates Hibernate's SQL-level @Filter on every repository method call, providing a designed-to-prevent-bypass isolation boundary independent of application code discipline. Second, a five-feature Gemini LLM integration pipeline with asynchronous thread-pool decoupling, empirically shown to reduce user-perceived AI latency by a factor of
7.8. Third, a real-time voice AI interface using a custom OpenAI-compatible streaming endpoint that routes Vapi.ai voice queries through Gemini without a commercial LLM subscription. Security evaluation confirmed correct enforcement of all isolation, authentication, and authorization mechanisms across eight adversarial scenarios. AI classification achieved a weighted F1-score of 0.86 on 150 labelled tickets, consistent with prior LLMbased triage literature. The system is fully containerised and cloud-deployed, demonstrating production viability and providing a replicable architectural reference for multitenant, AI-augmented SaaS platform development.

REFERENCES

[1] OWASP, "Broken Access Control," OWASP Top 10 Web Application Security Risks, 2021. [Online]. Available: https://owasp.org/Top10/A01_2021Broken_Access_Control/

[2] D. Chong and H. Hwang, "Multi-Tenancy in Cloud Computing: A Survey of Isolation Approaches," IEEE Access, vol. 8, pp. 114946–114963, 2020.

[3] T. Brown et al., "Language Models are Few-Shot Learners," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.

[4] P. Orosz and T. Farkas, "An Empirical Study of AI-Based Triage in Enterprise Support Tickets," in Proc. IEEE ICSOC, 2022.

[5] J. Wei et al., "Finetuned Language Models are Zero-Shot Learners," in Proc. ICLR, 2022.

[6] W. Weissman, S. Bobrowski et al., "The Multi-Tenant Architecture," Salesforce White Paper, 2005.

[7] C. Bauer and G. King, Java Persistence with Hibernate, 2nd ed. Manning Publications, 2015.

[8] S. Aulbach et al., "Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques," in Proc. ACM SIGMOD, pp. 1195–1206, 2008.

[9] J. Sun, Y. Zhang, and Q. Chen, "A Middleware Framework for Multi-Tenant Data Isolation in SaaS Applications," IEEE Trans. Cloud Computing, vol. 7, no. 2, pp. 432–445, 2019.

[10] R. S. Sandhu et al., "Role-Based Access Control Models," IEEE Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.
[11] Spring Framework, "Spring Security Reference Documentation," 2024. [Online]. Available: https://docs.spring.io/spring-security/reference/
[12] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in NeurIPS, 2022.
[13] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in EMNLP, pp. 3982–3992, 2019.
[14] B. Hancock et al., "Learning from Dialogue after Deployment," in Proc. ACL, 2019.
[15] Vapi AI, "Real-Time Voice AI Platform — Technical Overview," 2024. [Online]. Available: https://vapi.ai/docs
[16] Google, "Gemini API Developer Documentation," 2024.
[Online]. Available: https://ai.google.dev/docs
[17] Clerk, "Clerk Authentication and User Management Documentation," 2024. [Online]. Available:
https://clerk.com/docs
[18] S. Newman, Building Microservices: Designing FineGrained
[19] Systems, 2nd ed. O'Reilly Media, 2021.