IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

An Integrated Saas Framework For Smart IT Service Management: Combining AI Chatbots And Automated Asset Tracking

Vishal Abasaheb Borude

Department of MCA, JSPM University, Pune, Maharashtra, India

Guide: Dr. Faizur Rashid, JSPM University, Pune

***Abstract*—**IT service desks in growing organizations routinely absorb a disproportionate share of staff hours on requests that fall below the skill threshold of the engineers handling them. Password resets, printer connectivity, VPN drop-outs, and software access queries collectively account for a large fraction of inbound tickets in any mid-sized IT team. This paper presents Vertexa Vision, a web-based Software-as-a-Service (SaaS) platform that consolidates three operational functions—structured ticket management, real-time IT asset tracking, and keyword-driven chatbot assistance—into a single application sharing one data model. The system is built on the MERN stack: Next.js handles the frontend, Express.js on Node.js serves the REST API layer, and MongoDB Atlas provides persistent document storage. JSON Web Token (JWT) authentication enforces three-role access control across all API routes. The chatbot module processes incoming natural language queries through a pattern-matching NLP engine and resolves recognized requests without human intervention, targeting a deflection rate above 60% for standard tier-1 issues. A three-tier subscription model (Silver, Gold, Diamond) scales the platform from small teams of 50 users to deployments exceeding 500 concurrent users. System performance was validated against 25 functional test cases spanning authentication flows, ticket lifecycle transitions, chatbot response accuracy, and asset record integrity. A 30-day pilot with 12 volunteer users recorded an average ticket resolution time of 4.2 hours, dropping to 18 minutes for requests handled autonomously by the chatbot. API response times held below 400 ms at 100 concurrent virtual users. The framework provides a deployable, open-architecture alternative to commercial ITSM products whose licensing costs place them out of reach for smaller IT organizations.

***Index Terms*—**IT service management, SaaS, MERN stack, NLP chatbot, asset tracking, ticketing system, JWT authentication, role-based access control, MongoDB, Next.js

I. INTRODUCTION

IT departments face a recurring operational imbalance. Support request volumes grow roughly in proportion to headcount, but IT staffing does not. The typical response in organizations without a formal ITSM tool is a combination of shared email inboxes, messaging app threads, and spreadsheets—a set of channels that provides no consistent tracking, no SLA enforcement, and no way to correlate hardware problems with the tickets raised against specific assets.

Structured ITSM platforms exist to fill this gap, but the commercial leaders impose substantial entry costs. ServiceNow's ITSM module carries licensing fees that can exceed $100 per named user per month at enterprise tiers [1], a figure that rules it out for the majority of small and medium-sized enterprises (SMEs). Freshservice offers a lighter pricing model but requires significant configuration effort before workflows reflect an organization's actual support structure [2]. Neither product ships with a native chatbot layer integrated into the same data model as tickets and assets.

Open-source alternatives such as OTRS and osTicket address the cost barrier but lack built-in AI assistance and typically require on-premise hosting that raises operational complexity for teams without dedicated infrastructure staff. The result is that many SME IT teams continue operating with fragmented tooling long after the cost of that fragmentation—in engineer time and service quality—exceeds what a purpose-built platform would cost.

This paper describes Vertexa Vision, a unified ITSM platform developed as a research prototype targeting mid-market IT teams with 50 to 500+ users. The platform pursues three specific goals: first, to give end users a structured, trackable channel for submitting support requests; second, to give administrators a live inventory of

hardware and software assets with bidirectional links to open incidents; and third, to deflect common tier-1 requests through an NLP-driven chatbot before they enter the engineer queue.

The remainder of the paper is organized as follows. Section II surveys related work in ticket classification, IT chatbots, and asset management.

Section III

describes the system architecture across its three main modules. Section IV details the implementation decisions for each layer of the stack. Section V presents evaluation results from functional testing and a pilot deployment. Section VI concludes with planned extensions.

II. RELATED WORK

Research on IT support automation has historically treated ticketing, asset management, and conversational assistance as separate problems. The bodies of work that bear most directly on this paper are reviewed below.

A. Ticket Classification

Kumar et al. [3] applied a Random Forest classifier with TF-IDF feature extraction to a corpus of 12,000 historical support tickets from an enterprise IT environment. Their system achieved 87% categorization accuracy, correctly routing tickets to hardware, software, network, or access management queues. The work demonstrated that ML-based routing is viable at production scale, but the system offered no self-service resolution path—every classified ticket still required a human engineer. Asset management was outside the paper's scope entirely.

Rajpurkar et al. [11] surveyed NLP-driven service desk automation patterns and identified a recurring limitation: models trained on historical ticket text tend to degrade when an organization's technology stack changes, because the vocabulary of issues shifts with the tools in use. The paper recommended re-training cycles tied to infrastructure change events, a practice that is practical for large IT organizations but costly for smaller teams.

B. Conversational Agents for IT Support

Xu et al. [4] built a retrieval-based chatbot for a university help desk using BERT sentence embeddings to match incoming queries against a curated FAQ collection. The system handled approximately 70% of password reset and account access queries without human involvement. The key constraint was maintenance overhead: the FAQ database required continuous manual curation as policies and systems changed, and the system's accuracy fell noticeably when queries diverged from the style of the training corpus.

Chen and Liu [16] explored transformer-based incident classification in cloud operations, demonstrating that pre-trained language models can generalize across IT domains better than task-specific classifiers. Their work focused on classification rather than user-facing resolution, and the compute requirements for inference ruled out deployment on modest infrastructure.

IT Asset Management

Naik and Reddy [5] proposed an RFID-assisted hardware tracking system designed for campus-scale deployments. Physical asset movements were logged automatically as equipment passed through tagged checkpoints. Location accuracy was high, but the system had no interface connecting asset records to support incidents. An engineer responding to a hardware fault still had to consult a separate system for the device's service history.

Traditional approaches to software asset management, reviewed by Alzahrani et al. [6], rely on agent-based inventory scans that enumerate installed software on endpoints. These approaches capture what is installed but not the context of why—information that surfaces only when a software issue generates a support ticket.

C. Summary of Gaps

Across these bodies of work, the shared limitation is modularity: ticket systems do not know about asset records, and chatbots do not know about open tickets. Vertexa Vision addresses this by building all three functions on a common document store, where a user's asset assignments, ticket history, and chatbot interactions share a single identifier and can be retrieved together in one query.

III. SYSTEM ARCHITECTURE

A. Overview

Vertexa Vision follows a three-tier architecture: a browser-based client tier, a stateless REST API application tier, and a cloud-hosted database tier. All communication between tiers uses HTTPS with TLS 1.3. The application is deployed as a multi-tenant SaaS service, with tenant isolation enforced at the data layer through a tenantId field present on every document.

B. Role Model and Access Control

Three roles govern what each user can see and do within the system. End Users can submit tickets, view their own ticket history, and interact with the chatbot, but cannot access asset records or other users' tickets. Support Engineers can view and update all tickets in their assigned category queue and access asset records linked to a ticket's submitting user, but cannot modify system configuration. Administrators have unrestricted access to all tickets, the full asset inventory, user management, subscription configuration, and the chatbot knowledge base.

Role assignment is embedded in the JWT payload at login time. A middleware function applied to every protected route validates the token signature, checks expiry, and asserts that the role claim meets the minimum required for the requested operation. Failed

assertions return HTTP 403 with a structured error body.

C. Ticketing Module

A ticket submission captures: title, description, category (Hardware / Software / Network / Access), priority (Low / Medium / High / Critical), and the submitting user's identifier. The system generates a unique ticket ID using a combination of a timestamp prefix and a random 6-character suffix, which makes IDs both sortable and collision-resistant without a database sequence.

On submission, an event hook passes the ticket's category and a tokenized form of the description to the chatbot engine. If the engine returns a match above the confidence

threshold, a suggested resolution is appended to the ticket record and surfaced to the user before the ticket enters the engineer queue. Users can mark the suggestion as helpful and close the ticket immediately, or dismiss it and allow the ticket to proceed normally.

Status transitions follow a defined lifecycle: Open → In Progress → Pending User Response → Resolved → Closed. All transitions are timestamped and logged to an embedded audit array within the ticket document. This design avoids the need for a separate audit table and keeps the full ticket history accessible in a single document fetch.

D. Asset Management Module

Hardware assets are represented with the following fields: assetTag, make, model, serialNumber, purchaseDate, warrantyExpiry, assignedUserId, location, and condition. Software assets capture: licenseName, vendor, version, licenseKey, seatCount, assignedUserIds, and renewalDate.

When a support engineer opens a ticket, the interface fetches assets assigned to the ticket's submitting user via a MongoDB aggregation pipeline that joins the assets and tickets collections on userId. The result displays alongside the ticket detail, giving the engineer immediate context about the reporter's hardware configuration without navigating to a separate screen.

Warranty and license renewal alerts are generated by a scheduled job that runs nightly and flags records within 30 days of expiry. Alerts are written to a notifications collection and surfaced on the administrator dashboard.

E. NLP Chatbot Module

The chatbot engine uses a keyword-pattern matching model. A knowledgeBase collection in MongoDB holds response entries, each consisting of: a set of trigger keywords, a set of excluded keywords (to avoid false positives), a category weight, and a response template. At query time, the input string is lowercased and tokenized, stop words are removed using a fixed list stored in application memory, and remaining tokens are stemmed using a lightweight Porter stemmer [12].

Each knowledge base entry is then scored using the function: $\text{Score}(q, r) = [\Sigma\, w_i \times \text{match}(t_i, K_r)] \,/\, |T^i|$, where T^i is the processed token set of the query, K_r is the keyword set of response r, match() returns 1.0 for an exact stem match and 0.6 for a partial stem match, and w_i is the category weight for the matched keyword. The entry with the highest score above a threshold of
0.45 is selected. Below threshold, the chatbot creates a ticket on the user's behalf and confirms this in its reply.

The knowledge base was seeded with 85 entries covering password resets, VPN connectivity, printer setup, software installation, account access, and hardware fault reporting. Entries were authored by a practicing desktop support engineer to reflect the vocabulary that real users employ when describing problems, rather than the formal language of documentation.

F. Subscription Tier Enforcement

Three subscription tiers control resource limits per tenant, as summarized in Table I. Tier limits are checked by middleware that reads the tenant subscription document before processing user creation or file attachment requests. Tenants approaching their user limit receive dashboard warnings at 80% and 95% utilization.

TABLE I
Subscription Tiers and Resource Limits

Tier	Max Users	Storage	SLA Response Target
Silver	50	10 GB	48 hours
Gold	200	50 GB	24 hours
Diamond	500+	Unlimited	4 hours

IV. IMPLEMENTATION

A. Frontend

The client application is built with Next.js 14 using the App Router pattern. Server Components handle data-fetching for the dashboard and ticket list views, reducing the JavaScript payload delivered to the browser and improving Time to First Byte [14]. Tailwind CSS provides utility-based styling without a separate CSS build step. Framer Motion animates panel transitions and modal entry/exit to provide visual continuity as users move between sections.

The chatbot interface renders as a collapsible panel accessible from any page in the application. WebSocket connections were considered for real-time updates but rejected in favour of short-poll requests at 10-second intervals, which simplified deployment by eliminating the need for sticky sessions or a WebSocket proxy.

B. Backend API

The Express.js API exposes four resource groups: /api/auth, /api/tickets, /api/assets, and /api/chatbot. Each route group has its own controller file, middleware chain, and Joi validation schema. Input validation runs before route handlers execute; invalid requests are rejected with HTTP 422 and a field-level error list without reaching the database layer.

Error responses follow a consistent envelope: { success: false, error: { code, message } }. This uniformity allows the frontend to handle all error conditions with a single interceptor rather than per-endpoint error logic, in keeping with REST architectural constraints [9].

C. Database Design

MongoDB Atlas was selected for its document model, which maps naturally to the nested structure of ticket audit logs and asset assignment histories, and for its managed operations—backups, patching, and failover are handled by the platform without operator intervention [13]. Six collections are used: users, tickets, assets, knowledgeBase, notifications, and tenants.

Indexes were created based on observed query patterns during development: compound indexes on (tenantId, status, createdAt) for ticket list queries, and on (tenantId, assignedUserId) for asset lookups. These indexes reduced

ticket list query times from an average of 840 ms to 187 ms on the test dataset.

D. Authentication and Security

User passwords are hashed with bcrypt at a cost factor of 12 before storage [17]. JWT tokens are signed with HMAC-SHA256 using a 256-bit secret loaded from an environment variable at startup [8]. Token expiry is set to 24 hours. MongoDB Atlas connections use TLS 1.3 and are restricted to the application server's IP address through Atlas network access rules, preventing direct database access from outside the application tier.

V. RESULTS AND DISCUSSION

A. Test Environment

Functional testing was conducted on a development server running Node.js 20 LTS. Load testing used a MongoDB Atlas M10 cluster (2 vCPUs, 2 GB RAM). Concurrent user load was simulated with Apache JMeter using virtual user threads ramped from 10 to 100 over a 5-minute ramp period with a 10-minute steady-state hold.

B. Functional Test Results

25 test cases were executed across four categories. Results are summarized in Table II. The single failing case involved a compound query—"my laptop's VPN is not connecting and I also need a license for Adobe Acrobat"—where the engine returned only the VPN response. The scoring function selects a single best-match entry; multi-intent queries that span distinct knowledge base entries are not currently handled. This is a known architectural limitation of single-response retrieval.

TABLE II

Functional Test Results by Category

Category	Cases	Pass	Fail
Authentication and RBAC	6	6	0
Ticket Lifecycle	9	9	0
Asset Management	5	5	0
Chatbot Resolution	5	4	1
Total	**25**	**24**	**1**

C. Chatbot Evaluation

The chatbot was evaluated against a held-out set of 200 queries collected from volunteer users who were asked to describe common IT problems in their own words without coaching. Queries matched above threshold: 142 (71%). Queries below threshold due to genuinely novel issues: 34 (17%). Queries below threshold due to phrasing variations of known issues: 24 (12%).

The 24 phrasing-variation failures indicate that expanding keyword sets for existing entries—without adding new entries—would improve coverage. For example, queries about "internet not working" did not consistently match entries keyed on "network connectivity" because neither token survived the stop-word removal and stemming pipeline in a form that scored highly against the other.

D. Resolution Time

A 30-day pilot with 12 volunteer users generating synthetic support requests produced the following resolution time data: overall average resolution time was 4.2 hours; tickets resolved via chatbot suggestion accepted by the user averaged 18 minutes; tickets requiring engineer intervention averaged 6.8 hours. The chatbot-assisted path is approximately 22 times faster than the engineer path for the issues it can handle. The overall average is pulled upward by engineer-path tickets, which represent the majority of volume in this small pilot.

E. API Performance

At 100 concurrent virtual users, all endpoints remained below the 2-second threshold defined in the test plan. Detailed latency measurements are shown in Table III. The chatbot query endpoint showed the lowest latency because it operates entirely in application memory after the knowledge base is loaded at startup, avoiding a database round-trip for the scoring computation.

TABLE III

API Response Times at 100 Concurrent Virtual Users

Endpoint	Mean Response (ms)	95th Percentile (ms)
POST /api/tickets	312	541
GET /api/tickets	187	334
GET /api/assets	198	361
POST /api/chatbot/query	94	152

F. Discussion

The results confirm the core thesis: combining ticketing, asset management, and chatbot assistance in a shared data model produces measurable operational improvements without requiring expensive infrastructure. The 71% chatbot deflection rate is comparable to results reported by Xu et al. [4] using a BERT-based retrieval approach, achieved here with a keyword-scoring engine that requires no GPU and no model training cycle.

The primary limitation is the chatbot's inability to handle novel phrasing and multi-intent queries. This limitation is inherent to keyword matching and would require a language model to address properly. The trade-off is deliberate: keyword-based responses are fully auditable, can be updated by a non-technical administrator, and add no inference cost per query.

CONCLUSION

This paper presented Vertexa Vision, a SaaS ITSM platform that integrates ticket management, IT asset tracking, and NLP-assisted self-service into a single application. The system passed 24 of 25 functional test

cases, achieved a 71% chatbot deflection rate on the evaluation query set, recorded an 18-minute average resolution time for chatbot-handled requests versus 6.8 hours for engineer-handled requests, and sustained sub-400 ms API response times at 100 concurrent users.

The contribution is architectural rather than algorithmic: placing tickets, assets, and chatbot interactions in a common data model with a shared user context eliminates the cross-tool context-switching that consumes engineer time in organizations using separate products for each function.

Future work will pursue two directions. First, replacing the keyword-scoring engine with a quantized small language model would address multi-intent queries and phrasing-variation failures without the GPU infrastructure requirements of full-scale LLM deployment. Second, an API integration layer connecting to hardware procurement platforms would automate asset record onboarding, removing the manual data entry step that currently limits inventory accuracy at initial deployment.

REFERENCES

[1] ServiceNow, Inc., "ServiceNow IT Service Management—Product Overview," Technical Datasheet, 2024.

[2] Freshworks Inc., "Freshservice ITSM Product Guide," Product Documentation, ver. 2023.4, 2023.

[3] S. Kumar, R. Patel, and M. Singh, "An Intelligent IT Service Management System Using Machine Learning and NLP," in Proc. IEEE Int. Conf. Computing, Communication and Automation (ICCCA), Greater Noida, India, 2022, pp. 45–51.

[4] Y. Xu, L. Wang, and J. Chen, "A BERT-Based Chatbot for University IT Help Desk Automation," IEEE Access, vol. 10, pp. 82345–82356, 2022, doi: 10.1109/ACCESS.2022.3194872.

[5] S. Naik and P. Reddy, "RFID-Enabled IT Asset Tracking System for Campus Networks," in Proc. IEEE Int. Conf. RFID Technology and Applications (RFID-TA), Delhi, India, 2021, pp. 119–124.

[6] A. Alzahrani, H. Alotaibi, and M. Alshehri, "Evaluating Open-Source ITSM Platforms Against ITIL v4: A Comparative Study," IEEE Trans. Services Comput., vol. 16, no. 2, pp. 1102–1115, Mar. 2023.

[7] T. Bhatt and A. Sharma, "Role-Based Access Control Implementation in Multi-Tenant SaaS Applications," in Proc. IEEE Int. Symp. Software Reliability Engineering Workshops (ISSREW), 2022, pp. 78–83.

[8] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force RFC 7519, May 2015.

[9] R. T. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," IETF RFC 7230, Jun. 2014.

[10] D. Zhao, H. Lu, and Q. Wang, "Performance Benchmarking of Node.js REST APIs Under Concurrent Load," in Proc. IEEE Int. Conf. Cloud Engineering (IC2E), San Francisco, CA, USA, 2021, pp. 211–218.

[11] A. Rajpurkar, T. Goel, and K. Gupta, "NLP-Driven Service Desk Automation: Design Patterns and Pitfalls," IEEE Software, vol. 40, no. 1, pp. 34–41, Jan./Feb. 2023.

[12] M. F. Porter, "An Algorithm for Suffix Stripping," Program: Electronic Library and Information Systems, vol. 14, no. 3, pp. 130–137, 1980.

[13] MongoDB, Inc., "MongoDB Atlas Architecture and Security Guide," Technical Whitepaper, rev. 2023.

[14] Vercel, Inc., "Next.js 14—App Router Architecture," Official Documentation, 2023. [Online]. Available: https://nextjs.org/docs

[15] AXELOS Ltd., ITIL 4 Foundation: IT Service Management. London, U.K.: TSO, 2019.

[16] H. Chen and W. Liu, "Automated Incident Classification in Cloud Operations Using Transformer Models," IEEE Trans. Network Service Manage., vol. 19, no. 3, pp. 2841–2853, Sep. 2022.

[17] A. Nair and P. Thomas, "Evaluation of bcrypt for Long-Term Password Storage Security," in Proc. IEEE Int. Conf. Cybersecurity and Privacy (ICCP), 2020, pp. 89–94.

[18] O. Hayder, Practical Node.js: Building Real-World Scalable Web Apps, 2nd ed. New York, NY, USA: Apress, 2017.

[19] W. Stallings, Cryptography and Network Security: Principles and Practice, 8th ed. Hoboken, NJ, USA: Pearson, 2022.

[20] F. Rashid and V. A. Borude, "Architectural Considerations for Integrated ITSM SaaS Platforms in SME Environments," JSPM University Research Proceedings, vol. 3, pp. 15–22, 2026.