Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TenantX: Enterprise-Grade Multi-Tenant SaaS Platform with Dynamic RBAC and Configurable Workflow Engine

Dr. Mahmood Ali Mirza
[1]Assistant Professor, Department of Computer Science and Engineering, Krishna University College of Engineering and Technology, Krishna University,

G. Purna Ajay, K. Hareesh , B. Brahmaiah, J. Rohith
[2] Student, Department of Computer Science and Engineering, Krishna University College of Engineering and Technology, Krishna University, Rudravaram, Andhra Pradesh, India.

***Abstract* - The situation where modern businesses are more dependent on the Softwareas-a-Service (SaaS) platforms to provide scalability and centrally is taking a new shape. managed digital solutions. However, building enterprise-grade With multi-tenant systems, essential challenges such as are presented. immobile isolation of tenant, dynamic access control of tenant, workflow regulatory, subscription governance and automation. compliance. Conventional architectures are usually reliant on. rigidly defined roles, fixed authorization policy, and restricted. auditing functionality, and hence, minimizing flexibility and scalability over the long term. There is no common architectural. authentication that is concurrently addressed by a framework. configurable authorization, embedded process, boundaries. orchestration, monetization governance form is a considerable difference in modern SaaS design. This document introduces a multi-tenant based on enterprise grade, TenantX. Cloud-based solution to solve these issues with the help of a homogenous and secure structure. The system proposes a dualcontext JWT authentication boundary separating SaaS. access to administrative functions of tenant-level activities, guaranteeing. strict multi-domain isolation. A dynamic active Role-Based. RBAC engine allows role that is tenant-specific. and authorisation without code changes. An built-in template-based workflow engine is multistep approval, lifecycle management and role-controlled. task routing. Monetization as a subscription is imposed. with a centralized quota validation scheme via HTTP. 402 signaling. Security is systematically assessed by means of the. A Zero Trust-aligned threat model is the STRIDE. principles. Django REST Framework is used to implement it. The performance under test are PostgreSQL, Redis, and React. simulated enterprise load (100 tenants, 10,000 users, 50,000 workflow requests) provides P95 API latency of less than 200 ms. and close-to-linear horizontal scalability.**

***Keywords* - Multi-Tenancy, Role-Based Access Control (RBAC), JSON Web Tokens(JWT), Workflow Automation., Zero Trust Architecture, SaaS Subscription Billing, STRIDE Threat Modeling**

I. INTRODUCTION

A. Background and Motivation

Software-as-a-Service (SaaS) has now become the new paradigm of software delivery in contemporary enterprise computing. SaaS services, unlike conventional on-premise systems are centrally stored, subscriptions and use web interfaces, so the end-user does not need to operate infrastructure. This model allows quick deployment, ongoing software, updates, and infinite scalability and predictability of the costs. The modern SaaS is designed architecturally based on cloud-native design such as stateless API design, centralized authentication, distributed caching, and horizontally-scaling databases.

With SaaS platforms being expanded to accommodate the needs of multiple organizations across a common infrastructure, they need to facilitate secure sharing of resources, and ensure very strong logical isolation of tenant information. This is the most common multi-tenant architecture (also referred to as multi-tenancy), which can be conducted in three main strategies, which include: separate databases to each tenant (strong isolation, high operational cost), sharing databases but with different schemas (balanced approach), and sharing databases but with shared schema differentiated by tenant identifiers (maximum scalability through rigorous logical enforcement). The goals behind multi-tenancy are efficiency in the economy, lower operational overheads, centralized (or central) control of upgrades, and the need to be able to handle increasing customer populations without corresponding increases in infrastructure.

B. Enterprise Challenges

Although multi-tenancy has its benefits, there are various architectural and functional problems in SaaS systems at the enterprise grade. Isolation of tenants needs to be imposed at every layer of authentication, authorization as well as the access to data- tenants- poised appropriate isolation mechanisms may provide cross-tenant accessibility to information with disastrous security failures and penalties. Dynamic access control is necessary since the conventional systems where hardcoded positions are used like Admin or Manager need to be changed every time the organizational policies change. The automation of workflows is seen as becoming more and more of a central element of enterprise

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

activities; organizations in need of multi-step approval of financial approvals, leave, and compliance verifications. Inter-domain authentication poses some special complexity since platform administrators, and user-level identities need to manage identity environments which are firmly separated. Monetization via subscription opens governance pitfalls where monotony in resource allocation is necessitated. The audit and compliance regulations require the use of logging that is tamper resistant in order to reinforce the regulatory systems. Lastly, scalability limitations entail the ability to serve thousands of concurrent users and high-latency.

C. Research Objectives

The main aim of the research is to establish and deploy a multi-tenant SaaS enterprise-level platform with the ability to deliver safe tenant separation, dynamic permission, workflow automation, and resource regulation by subscription, all in a solitary system composition. The research aims to accomplish this by: (1) designing a dual-domain authentication model that separates SaaS administrators and tenant users; (2) deploying a dynamic RBAC engine that can dynamically configure its roles and permissions; (3) creating a configurable multi-step workflow engine that includes structured lifecycle operations; (4) strictly isolating tenants at API and database levels; (5) implementing subscription-based quota enforcement mechanisms

D. Paper Contributions

The paper provides the following important contributions to the design of secure multi-tenant SaaS architecture:

- An administrative-to-tenant SaaS micro-architecture that isolates administrative operations and tenant-level operations, preventing misuse of cross-domain tokens as well as enhancing logical isolation.
- Dynamic Runtime RBAC Engine The ability to enable tenants to create, modify and assign roles without changes in code making it more adaptable and scalable.
- A Workflow Engine with Templates which will be able to support the configurable approval processes in form of multiple steps included to the SaaS itself.

All: Authorized Unified billing and quota enforcement architecture, which supports consistent plan-based resource management both in the backend and the frontend over HTTP 402 signaling.

- A Formal STRIDE-Based Security Analysis that is a methodical analysis of threats such as spoofing, tampering, repudiation, information disclosure, denial of service and privilege escalation.
- Experimentally verified, sub-200 ms P95 latency, and near-linear horizontal scalability with simulated enterprise workloads.

II. RELATED WORK

Enterprise grade multi-tenant SaaS platform design cuts across various areas of research, such cloud-based multi-tenancy models, formal theories of access control, workflow orchestration, monetization via subscriptions and Zero Trust security models. Although the existing literature has a lot of information to offer to each field separately, comprehensive architectural models that would bring the diverse parts together in a secure and scalable SaaS developed set-up are scarce.

A. Multi-Tenant Architectures

Multi-tenancy has been accepted as an essential architectural concept in cloud computing. Kumar et al. [1] categorize the multi-tenant database architectures into three categories which include, separate databases per tenant, shared databases with separate schemas and shared databases with shared schema with tenant identifiers. Shared-schema approach is most efficient and has a high level of horizontal scale but demands strict logical isolation between application and query layers. Most recent studies [2] are not limited to database segregation, but they also encompass application-layer segregation, API gateway authentication, scope of identity-bound requests, and namespace segmentation in Kubernetes. Several studies stress that infrastructure isolation is not a viable technique; to implement multi-tenancy, enforcement is necessary across authentication, authorization, business logic and data access layers. One long-standing gap found in implementations is that the separation between platform-level administrative domains and tenant-level operational domains is not formally enforced by the authentication layer, solely between the authentication layer and the upper-layer affecting user authentication, which is explicitly defined by the dual-context JWT authentication protocol used by TenantX.

B. RBAC and Access Control Models

Role-Based Access Control (RBAC), which was introduced in the standards of NIST formally, is the most adopted authorization model in the enterprise systems [3]. Project RBAC uses user-role assignments and role-to-permissions map when making access decisions. Other variants like Hierarchical RBAC add role inheritance and Constrained RBAC adds separation-of-duty policies. Although traditional RBAC systems have these advantages, these systems assume a fixed role definition, as hard-coded within central systems, and does not support per-tenant governed environment in dynamic SaaS environments. Attribute-Based Access Control (ABAC) is a federation of RBAC and attribute-based context-sensitive authorization but with greater computational demands and complexity in policy management. Adeyinka et al. [4] describe the relevance to use RBAC with the simplifying idea of Zero Trust in multi-tenant cloud systems and state that a mixture of role hierarchies based on structure and tenant-based validation diminishes the risk of privilege escalation considerably. TenantX is in line with dynamic RBAC theory by having a tenant scoped, runtime configurable engine where authorization decision is verified on each request with token based identity claims.

C. Workflow Automation Systems

Automation of workflow has developed out of the past Business Process Management (BPM) systems to embedded orchestration models incorporated in the SaaS offerings. BPM engines implemented on the BPMN standards have very rich modeling features such as event-driven triggers, parallel gateways, and risky decision trees but demand dedicated infrastructure and considerable operational

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

overhead [5]. New SaaS-oriented engines focus on the requirements of such lightweight template-based process definition as multi-step approval routing, state machine transitions, and role-based task assignment. An interesting research direction is the direct integration of workflow logic inside application layers instead of using external orchestration engines, thus resulting in less latency, and allowing easier deployment. Nevertheless, a large number of embedded solutions do not support native multi-tenancy or have global workflow definitions that cannot be customized by each tenant. In TenantX, there is a tenant-configurable, template-based workflow architecture, based on deterministic state transition theory, where all definitions and operational instances are tenant-scoped.

D. SaaS Billing and Zero Trust Security

The economic foundation of SaaS platforms is subscription-based monetization, but the architectural mechanisms to enforce the latter are under-researched. Billing in many systems is a marginal system that is not directly linked to the logic of operations, and this allows it to be bypassed [1]. New strategies promote API-enforcement of quotas and previous authoritative validation of resources prior to resource creation. The key principles in Zero Trust Architecture (ZTA) formalized in NIST SP 800-207 focus on sustaining continuous verification and least-privilege access instead of trust models situated at the perimeter [6]. Paul et al. [7] assert stateless JWT-based authentication is straightforward to use with Zero Trust since it can verify per request, and it does not maintain state between requests. Owning to the fact that a smaller number of studies report on systematic integration of Zero Trust in tenant-scoped SaaS systems, Makinde et al. [8] refer to a substantial literature gap. TenantX fills these gaps with combined server-authoritative billing enforcement and formal STRIDE-based security analysis and Zero Trust validation imposed on each request, offering a consolidated architectural structure, which is currently considered in isolation by existing systems.

III. PROBLEM STATEMENT

The contemporary enterprise software systems should be able to accommodate a variety of organizations, sophisticated managing approvals, adaptable access control models, regulation approvals mandatory, and high number of users. Nevertheless, a lot of the legacy web applications were not initially built with multi-tenancy, dynamic authorization and performance scalability in mind. These systems have structural flaws such as single-tenant deployment that increases the infrastructure costs and complexity of centralized updates, hard-coded role models in source code that require redeployment to change any policy; lack or insufficiency of tenant isolation at authentication and database layers that exposes cross-organizational data; weak or non-existent audit functionalities that impair forensic reliability and regulatory compliance; missing or limited workflow automation that requires manual

A coherent, enterprise-ready, multi-tenant SaaS architecture, which imposes strict tenant isolation at all levels of the system, integrating dynamic runtime-configurable RBAC, without having to redeploy application functions, a programmable multi-phase workflow model built-in into the platform, decoupled horizontal scalability of stateless, and smart caching are therefore a compelling requirement. TenantX is a specially developed solution to the problem and it is based on an integrated formally structured and security-first architecture.

IV. SYSTEM OVERVIEW

TenantX is a safe, customizable, business-grade multi-tenant SaaS solution uniting authentication, dynamic permission, workflow automation, subscription control, and audit logging into an all-encompassing construct. The system is designed to trade scalability, isolation, maintainability and extensibility without compromise of strict compliance to Zero Trust principles. On the architectural side, TenantX is designed as a modular monolith where domain modules are logically partitioned, and deployed together as one in a shared running environment, simplifying the distributed system while maintaining a sensible separation of concerns and allowing its future microservice decomposition should it be necessary.

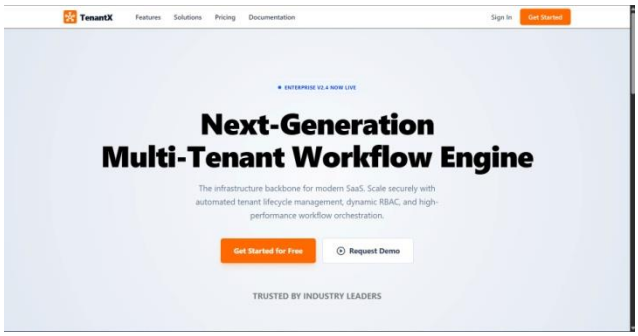

Fig. 1. TenantX Enterprise Multi-Tenant Workflow Engine — Landing Page

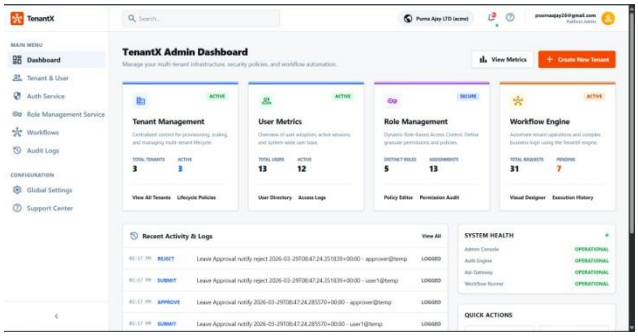

Fig. 2. TenantX Admin Dashboard — Centralized Management Hub

The architecture of TenantX is built in 5 logical layers: (1) Presentation Layer, React-based browser interface, user interaction, and secure token management; (2) API Layer, Django REST Framework, which is a gateway to perform request validation and routing to domain; (3) a layer One of the core architectural changes is the Dual Platform Model that isolates the SaaS Platform Layer and the Tenant Platform Layer and enforces JWT namespace, ensuring that cross-domain token reuse. The stack of technology is React + Vite and TanStack Query, Python/Django REST Framework, PostgreSQL 15, Redis 7 and Gunicorn/ Nginx is cloud-neutral and can run on AWS, Azure and Google Cloud.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

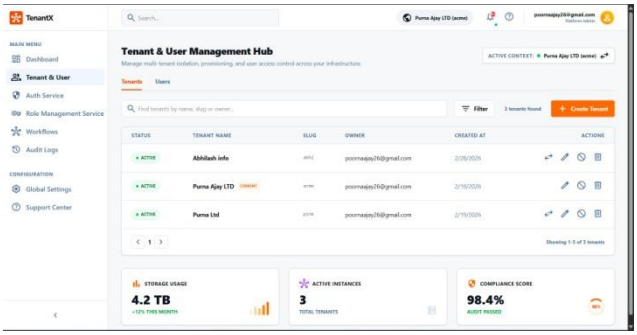

Fig. 3. Tenant & User Management Hub — Multi-Tenant Lifecycle View

V. SYSTEM ARCHITECTURE

A. Frontend Architecture

The frontend is built with React and a Vite-based build system and adheres to a strictly layered, component-oriented pattern of architecture. UI elements such as pages, layouts and modal interfaces are there just to render and engage with the user. They communicate with a custom hooks layer, which contains reusable business logic (authentication processing, role lookups, etc.). The service abstraction layer is closer to the hooks layer and contains all the API interactions in service modules of Axios-based service modules, each corresponding to a domain (authentication, tenant management, RBAC, workflow, billing). TanStack Query is used to do global server-state synchronization with deterministic cache management, background refetching, request deduplication and optimistic updates. An event bus, a centralized location, receives signals of HTTP 402 violation of quotas and ends up broadcasting prompts of upgrade throughout the system. Authentication is will be on a stateless model, and the tokens (access and refresh) will be stored in the memory and in the HttpOnly cookies respectively with short-lived (7-minute TTL) and long-lived (5-day TTL) time calibration respectively in order to avoid XSS threats.

B. Backend Architecture and Validation Pipeline

It uses the domain-driven modular structure and Django REST Framework to implement the backend. The fundamental areas that are implemented as independent application modules on the same runtime environment include platform user management, tenant management, RBAC enforcement, workflow orchestration, billing validation, and audit logging. All authenticated requests are authenticated through a sequence of checks in strict sequence: (1) JWT signature and expiration checking, (2) user-type domain checking (SaaS vs. tenant setting), (3) verification between the tenant identifier in token and request URL parameters, (4) RBAC permission checking against the necessary module.action pair, (5) billing quota checking (resource creators), (6) transactional database JWT verification and Redis blacklist checks are done in O(1) time, RBAC evaluation is done with O(R + P) complexity (R = assigned roles, P = permissions per role), and workflow state check is done with O(S) complexity (comparative to approval steps).

C. Database and Caching Architecture

PostgreSQL 15 is used by a shared schema multi-tenant model with the shared database used by TenantX. Each tenant-scoped table contains an explicit tenant_id column and all ORM queries enforce requirements on crossing tenant data that are based on verified request context, avoiding exposure of cross tenant data at data access layer. Composite B-tree indexes where tenant_id is the leading column are used such that one O(N) scan of the full table becomes an O(log N) index lookups. Tenant isolation is a defense-in-depth feature of optional row-level security (RLS) policies work mainly by the database engine. Redis is the distributed caching and coordination store of role-permission mappings (10-min TTL), listing of organizations (5-min TTL), billing usage counters (2-min TTL), and JWT blacklists (token lifetime). Nginx load balancing Stateless deployment architecture makes horizontal throughput scaling to near-linear levels, with empirically tested use at around 380 req/s per application node.

VI. AUTHENTICATION AND AUTHORIZATION

A. Dual JWT Context Model

The basic security of TenantX is based on authentication and authorization. In contrast to traditional SaaS applications based on a single identity context, which provide cohesive identity at the administrative and operational tiers, TenantX suggests a Dual JWT Context Model in which identity at the administrative level is distinct from identity at the operational level, specifically the SaaS and tenants, respectively. This formality avoids cross-context token abuse which is a typical privilege escalation path in single-domain SaaS designs.

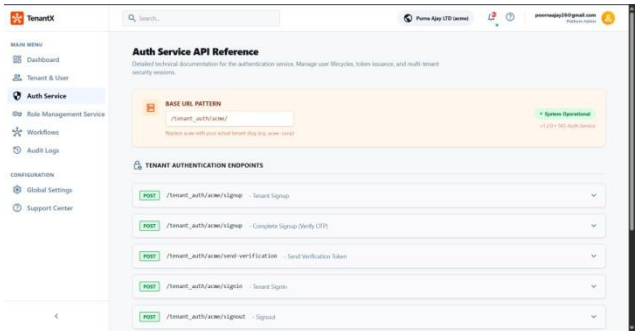

Fig. 4. Auth Service API Reference — Tenant Authentication Endpoints

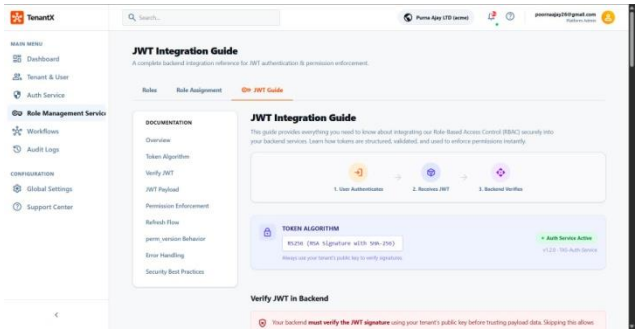

Fig. 5. JWT Integration Guide — Token Algorithm and Verification Flow

On authentication of a platform administrator, the system emits a SaaS scoped JWT with claims: user identifier, active organization identifier, token expiration date and time, a unique token identifier (JTI), and user type labelled as saas-

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

user. Such tokens can only be used with SaaS-level endpoints that handle onboarding of organisations, subscription, plan setup and global metrics. On the other hand, tenant users authenticate using tenant-scoped routes and obtain a tenant-specific JWT which carries the tenant user identifier, tenant identifier, assigned role list, expiration timestamp and user-type with the label tenant-user. These tokens can only be valid in the related tenant namespace. Test identifiers (JTIs) are considered to be stored in Redis, allowing them to be quickly revoked during a logout, by recognizing suspicious activity. Validating the signature (HMAC-SHA256), checking expiration, Redis lookups in blacklists and extracting claims are all O(1) operations.

One of the worst dangers of multi-tenant architecture is cross-tenant data leakage. Multi-layer isolation is enforced by tenantX: authentication layer checks tenant_id against the URL slug, application layer with mandatory ORM filtering on the basis on tenant-id blocks unauthorized access to data, optional database layer(s) using the PostgreSQL RLS to prevent bypass via direct queries, and the monitoring layer wherein unauthorized access attempts are logged to be examined during forensic investigations. When a tenant identifier in a JWT does not match the requested namespace, the system instantly responds with the HTTP 403 and logs the event. Rate limiting secures authentication endpoints with brute force attacks via Redis-supported per-user request counters that have customizable limits.

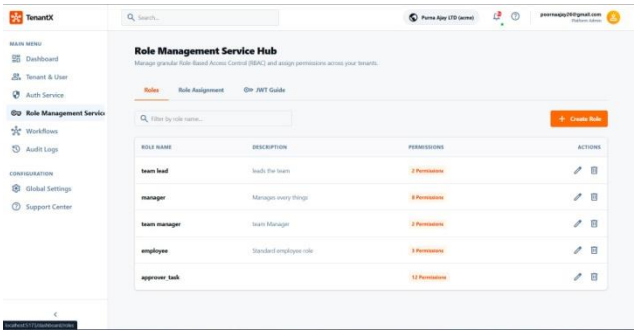

Fig. 6. Role Management Service Hub — Dynamic RBAC Configuration

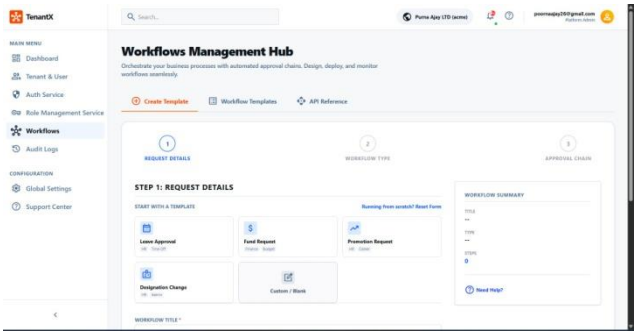

Fig. 7. Workflow Management Hub — Template Creation Interface

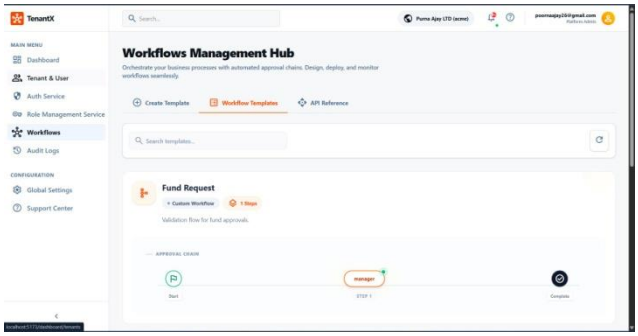

Fig. 8. Workflow Templates — Approval Chain Visual Designer

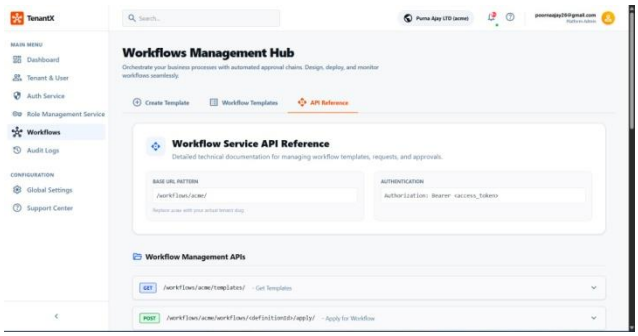

Fig. 9. Workflow Service API Reference — Base URL Pattern and Endpoints

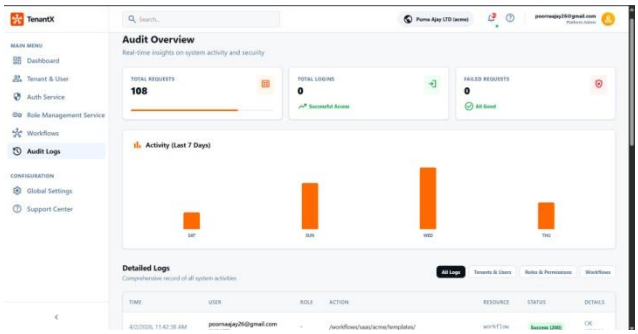

Fig. 10. Audit Overview — Real-Time Activity and Security Insights

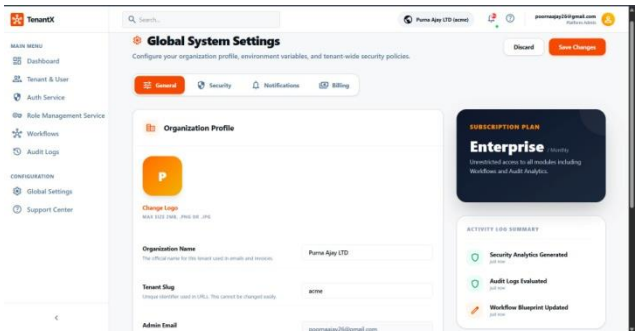

Fig. 11. Global System Settings — Organization Profile and Subscription Plan

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

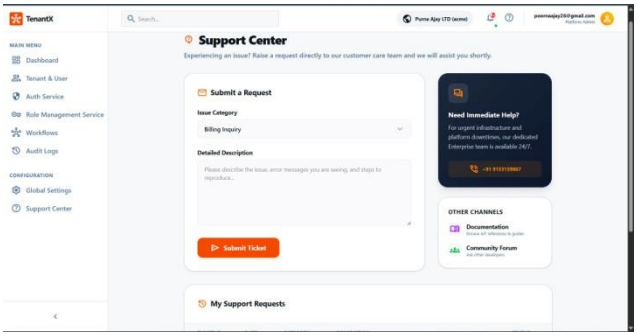

Fig. 12. Support Center — Enterprise Customer Care Portal

VII. SCALABILITY AND OPTIMIZATION

A. Horizontal Scaling Strategy

Scalability Enterprise grade multi-tenant SaaS platforms must support scalability as a key feature. In tenantX, a stateless backend architecture has also been used which allows several instances of the application to run simultaneously behind a load balancer. Since authentication involves the use of JWT tokens to verify a token and does not use server-side storing session information, the state of the user is not stored on any application node in memory, and any available node can fulfill a request without having to be request-session affined. The Redis layer makes the blacklisting of tokens, role-permission associations and the quota counter in the form of a centralized store; which different nodes can access concurrently. PostgreSQL is the master database that has optional read replicas in the case of intensive read load. The auto-scaling groups of clouds are capable of dynamically adding nodes to the group when there is a burst in traffic without compromising the integrity of authentication.

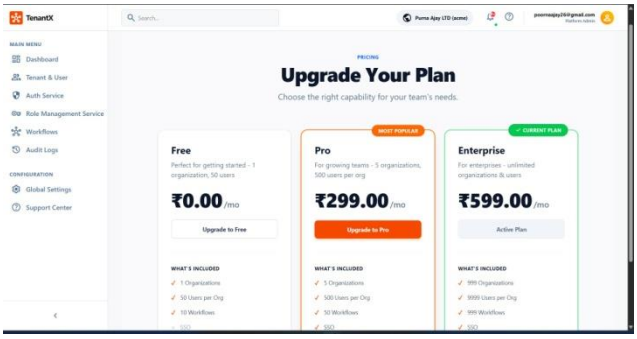

Fig. 13. TenantX Pricing — Free, Pro, and Enterprise Subscription Plans

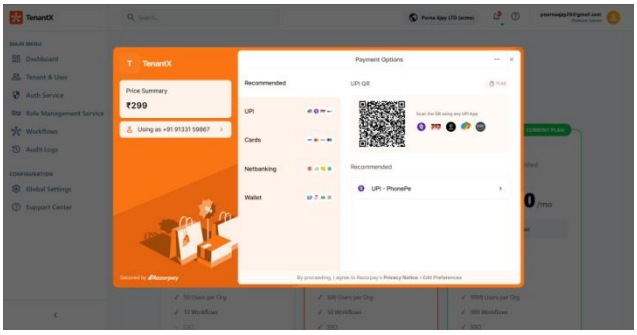

Fig. 14. Razorpay Payment Gateway Integration — UPI and Card Payment Options

B. Database Indexing and Query Optimization

To ensure low query latency in a multi-tenant system where the majority of queries are tenant-scoped, efficient indexing is needed. In tenancyX, composite B-trees are used where the tenant key (tenant ID) is the primary key to achieve this, meaning that when querying the database, the search scope is immediately limited to the tenant partition being targeted and the process is reduced beyond O(N) scans of the entire table to O(log N) lookups in the index. In addition to indexing, TenantX can optimize the behavior of ORM queries to remove N+1 query issues. Foreign key relationships are settled with select_related() resulting in reduction of repeated queries to associated entities. Many to many relationship: prefetchrelated() is used to batch-retrieve related objects. In scenarios where dashboards need read-only data, predominantly counts and common metrics are saved in Redis to minimize the number of computations required during peak usage.

TABLE III: Composite Index Strategy in TenantX

Index Pattern	Optimized Query Type	Complexity Improvement
(tenant_id, status, created_at)	Workflow filtering and chronological sorting	O(N) → O(log N)
(tenant_user_id, is_active)	Active user lookup and session validation	O(N) → O(log N)
(tenant_id, email)	Unique user resolution and authentication	O(N) → O(log N)
(tenant_id, created_at DESC)	Recent activity retrieval for dashboards	O(N) → O(log N)

VIII. PERFORMANCE EVALUATION

A. Benchmark Setup and Dataset

A simulated performance at an enterprise scale on Apache JMeter 5.6 at a controlled cloud-based deployment environment of 4 vCPU / 8 GB RAM, Django REST Framework, Gunicorn (4 workers), PostgreSQL 15 and Redis 7 were used to do performance benchmarking. The data set included 100 tenants, 10,000 total users, 50,000 workflow requests, 3-5 roles per tenant and 2-4 steps per workflow. The test scenarios were repeated five minutes in duration and with concurrency levels slowly raised to watch system behavior at sustained load. Audited scenarios were a concurrent user login, organization listing, role retrieval and permission validation, submitting a workflow and approval, and retrieving the audit log.

B. API Performance Results

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TABLE IV: API Performance Under Concurrent Load

Endpoint	Concurrent Users	Avg Response (ms)	P95 Latency (ms)	Error Rate
/auth/signin	100	48 ms	89 ms	0.0%
/auth/signin	500	112 ms	198 ms	0.2%
/organization/list	100	23 ms	41 ms	0.0%
/rbac/roles	100	31 ms	58 ms	0.0%
/workflows/action	100	67 ms	124 ms	0.0%
/audit/logs	50	145 ms	287 ms	0.0%

Authentication delays grow in a linear manner in regard to concurrency, which is expected during CPU-bound verification traffic. RBAC validation has low overhead of 2-4 ms per request. even moderate-concurrency (under 130 ms) workflow transitions are below 130 ms. The mean querying ratio of audit logs has high latency (aggregation and pagination) and the larger the error rates will be around less than 0.2% at 500 simultaneous log-ins.

C. Throughput and Horizontal Scaling Analysis

TABLE V: Horizontal Scaling Results

Application Nodes	Max Throughput (req/s)	P95 Latency (ms)	Scaling Efficiency
1 node	~380 req/s	198 ms	Baseline
2 nodes	~740 req/s	184 ms	~94.7% linear
3 nodes	~1,090 req/s	171 ms	~95.6% linear

The near-linear scalability is established when more application nodes are added. Stateless authentication allows node independent operation with no session affinity. Redis caching minimizes traffic contention in databases and load balancing ensures even distribution of traffic. The analysis of overheads of security test proves that the total cost of the enforced per request (JWT validation of about 3 ms, RBAC check of 2-4 ms, Redis blacklists lookup 1 ms) is negligible in comparison with the overall requests processing time. A full 3 stage workflow cycle submission (~60 ms), intermediate approval (~65 ms), final completion (~70 ms) is less than 200 ms. Standard enterprise load simulation did not show any critical bottlenecks.

D. Threats to Validity

The performance testing utilizes synthetic workloads, and controlled data that might not adequately represent real-world patterns of access, bursts, or malicious usage. The performance of the clouds across different providers, different geographic locations, and different storage solutions can be different. The mix of requests and the distribution of tenants was held constant in the test, which can underperform or overperform the real-life situation, with respect to both latency and throughput. These are the aspects to be taken into account in generalizing the individual performance characteristics.

IX. COMPARATIVE ANALYSIS

In comparing the architectural contribution of TenantX to three representative system categories, i.e. traditional generic CRUD applications, Camunda (workflow-based BPM engine), and Keycloak (identity-oriented access management platform), this section makes a comparison.

Simple CRUD apps are single-tenant and have hard-built roles, no workflow automation is provided, no subscription billing integration is available and the activity log is rudimentary. Their architecture is inadequate in supporting enterprise B2B SaaS workflows where tenant isolation, customizable workflows, and integrated subscription controls need to be in place. Camunda offers dynamic BPMN orchestration based on powerful modeling and can be deployed as a stand-alone JVM based engine, however, it needs integration with external identity and authorization systems; multi-tenancy is not fully supported, and subscription support is out of scope of the design. Keycloak is identity-centric and lacks workflow orchestration, subscription billing enforcement and full business domain tenant isolation, but provides strong identity management and SSO based on OAuth2/OpenID connect.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

TABLE VI: Consolidated Feature Comparison Across System Categories

Feature	Generic CRUD	Camunda	Keycloak	TenantX
Multi-Tenancy	Not supported	Partial support	Realm-based (identity only)	Native shared-schema model
Dual-Domain Authentication	Not supported	Not native	Partial	Fully implemented
Dynamic Runtime RBAC	Hardcoded roles	External IAM required	Realm-scoped, limited	Tenant-scoped runtime RBAC
Multi-Step Workflow Engine	Not available	Advanced BPMN (separate service)	Not available	Embedded state machine
Subscription Billing	Not integrated	Not supported	Not supported	Fully integrated
Backend Quota Enforcement	Not available	Not available	Not available	HTTP 402 backend-authoritative
Append-Only Audit Logs	Minimal	Process-level logs	Event logs	Immutable tenant-scoped logs
Zero Trust Alignment	Limited	Partial	Partial	Explicitly implemented
Horizontal Scalability	Limited	Moderate	High	Near-linear cloud-native
Deployment Complexity	Low	High (JVM-based)	Medium	Moderate (Python/React)

Only the TenantX architecture provides an integrated, multi-tenancy dynamic RBAC, workflow automation, subscription enforcement, audit logging and Zero Trust-compatible architecture. The integration makes TenantX an all-encompassing reference architecture to the state-of-the-art enterprise multi-tenant SaaS platforms, to include gaps in all three targets of comparison at the same time.

X. INNOVATION CONTRIBUTIONS

The key innovation of TenantX is the design integration and formalization of authentication, authorization, workflow management, and monetization control with rigid tenant isolation requirements. Instead of dropping unfocused technical capabilities, TenantX brings a harmonious architectural framework within which these capabilities work within well-known security confines.

The Dual-Context Boundary Model of JWT clearly divides the authentication into two functional realms, which tackles the cross-context privilege escalation vulnerability of single-domain SaaS. The novelty is not only in the utilization of JWTs but formalization of domain-separated identity contexts with a middle-level that strictly validates boundary crossings at the border of identity-definition a formalization itself that is scarcely found in multi-tenant SaaS literature. The Virtual Tenant Context Switching scheme permits SaaS administrators to serve multiple organizations with no repeated authentication processes, constant time contextual priority model (URL slug → virtual override → JWT default) and without compromising administrative privileges and tenant-level access credentials. Decoupling between monetization governance and presentation logic The HTTP 402-Based Centralized Quota Enforcement Architecture decouples all event processing in a global event bus interceptor, capturing all HTTP 402 responses to backend quota validation, resulting in a uniform upgrade process, and removing unnecessary quota approval checks within UI components. Enterprise-level process orchestration with deterministic state machine transitions, multi-step role-governed approvals with tenant-configurable definitions, and with O(S) complexity is demonstrated by the Embedded Multi-Tenant Workflow Engine, which is tightly coupled with tenant isolation and RBAC. Lastly, the formal STRIDE-Based Security Modeling represents a reproducible, methodologically transparent security assessment model to use in future SaaS designs, combining Zero Trust validation with both defense-in-depth layering of JWT, RBAC and ORM filtering as well as optional database RLS.

XI. FUTURE WORK

TenantX can be further refined by a few extensions that can help it be more flexible, intelligent and scale-able. First, Hybrid RBAC -ABAC Integration would enhance the model of dynamic RBAC that exists currently with attribute-based decision making. Role-Permission mapping would not be the sole condition used to determine access, but may be combined with contextual attributes such as department, amount of workflow thresholds, geographic location or time-of-day restrictions, modeled as: Allow = Role Permission Check∧ Attribute Policy Check. This will allow controlled governance and minimise role allocations to the privileged.

Second, AI-Based Workflow Risk Scoring would add a supervised learning model (e.g., gradient boosting) to the deterministic workflow engine that learns on past workflow data to score or deny approval requests with normalized risk levels, identify high-risk financial approvals, recognize abnormal workflow data patterns, and convert reactive automation into the engine to adaptive governance. Third, Cryptographic Audit Hash Chaining would add features of tamper-evidence based on blockchain principles, where every entry in the audit corresponds to an SHA-256 hash of its

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

predecessor: Hash(n) = SHA256(Hash(n-1) + LogEntry(n)). Periodical anchoring the external trust anchors or blockchain networks would give audit guarantees of tamper-evident audits.

Fourth, Microservice Decomposition would entail transforming the modular monolith into domain-driven microservices - individual services that authenticate, use roles and permissions, orchestrate workflows, charge users, audit logs and send notifications that communicate with one another using event-driven message brokers and REST/gRPC APIs, allowing each domain to be scaled independently, isolate failures, and deploy flexibly. Fifth, OAuth 2.0 (SSO) or OpenID Connect and SAML 2.0 federation of external identities (e.g. Microsoft Entra ID or Google Workspace) would allow Bring-Your-Own-Identity (BYOI) deployment models, and increase the applicability of TenantX to large enterprise settings.

XII. CONCLUSION

This paper introduced tenantX, a multi-tenant SaaS platform that is enterprise-grade and that provides solutions to key issues involving isolation of tenants, dynamic authorization, workflow automation, subscription control as well as organized audit logging. The drawbacks of traditional systems are the structural constraints of hard-coded role models, poor tenant isolation, weak orchestration of workflows, and weakly bound family standards of payment. TenantX has specifically been designed to surmount these constraints by introducing a single, modular and cloud-native architecture bringing together identity, authorization, workflow lifecycle management, and monetization governance within a coherent design.

Dual-Context JWT Boundary Model of the system is a formal separation of SaaS administration affairs and the workflows of tenants that minimizes the likelihood of cross-context privilege escalation. Dynamic role and permission configuration in tight tenant scopes can be configured disagreement at runtime with the dynamic RBAC engine. Its template-based workflow engine offers predictable state transitions, and multi-step approval routing at deterministic O(S) algorithmic complexity. To authoritatively manage subscription at the backend level, HTTP 402 quota validation is implemented to prevent billing limits being circumvented at the client layer. Append-only audit logging enhances traceability and accountability, and the formality of the threat-based security analysis in the form of the STRIDE enables the verification of thorough threat mitigation, which is in line with the principles of Zero Trust.

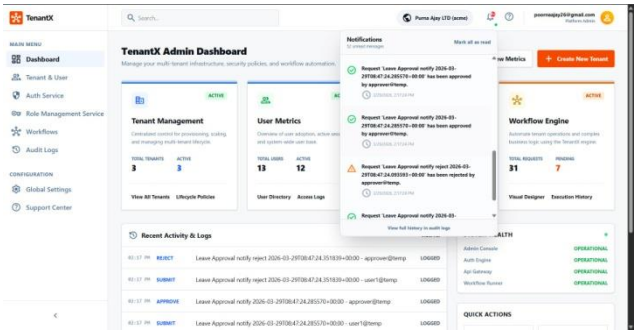

Fig. 15. Notification Panel — Real-Time Workflow Approval and Rejection Alerts

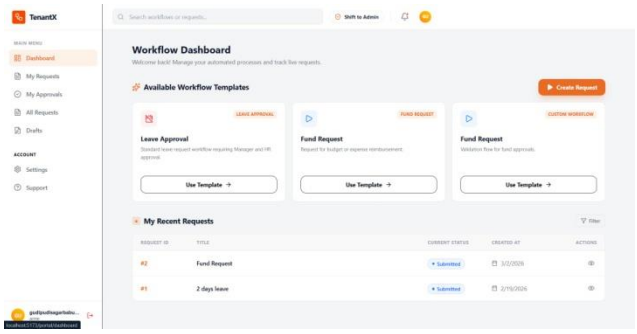

Fig. 16. Tenant User Portal — Workflow Dashboard with Available Templates

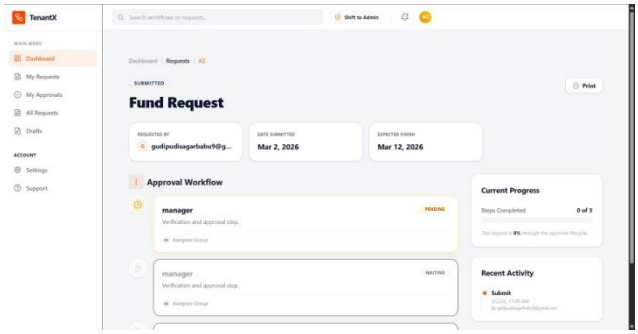

Fig. 17. Fund Request Detail — Approval Workflow Progress Tracking

Simulated enterprise performance testing (100 tenants, 10,000 users, 50,000 workflow requests) showed sub-200 ms P95 latency, scaling with the number of application nodes (approaching 1,090 req/s with 3 application nodes), and 68ms of total security overhead, per request (overall). These findings affirm that great isolation and authorization are synonymous with great performance. TenantX also illustrates that an expertly designed modular monolith can deliver the scalability and extensibility demanded of enterprise-level operations with none of the operational essentialism of microservices, as a viable and repeatable example of safe, scalable and governance motivated B2B SaaS systems that need rigorous multi-tenancy, customizable authorization and flexible workflow management.

Data Availability and Ethics Statement

In this study no actual customer data was employed. Synthetic test datasets or internal test datasets were drawn to run all experiments. To reduce exposure of personally identifiable information, the system is implemented to handle audit data under stringent access control and retention policies.

ACKNOWLEDGMENT

The authors would like to thank Krishna university honorable vice chancellor prof. Ramji koona for his endless support. We also like to extend our thanks to Krishna university start-up and innovation cell for their support in providing computing and other resources.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 04 , April - 2026

REFERENCES

[1] R. Kumar et al., "Multi-Tenant SaaS Architectures: Design Principles and Security Considerations," ResearchGate, 2024. [Online]. Available: https://www.researchgate.net/publication/391673039

[2] A. Rajuroy, "Blockchain-Assisted Secure File Sharing in Multi-Tenant Environments," ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/392363206

[3] D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access Control," Proc. 15th NIST-NCSC National Computer Security Conference, pp. 554–563, 1992.

[4] A. Adeyinka et al., "Zero Trust Architectures in Multi-Tenant Cloud Platforms: A Role-Based Access Control Reinforcement Framework," ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/393052657

[5] M. Weske, Business Process Management: Concepts, Languages, Architectures, 3rd ed. Berlin: Springer, 2019.

[6] NIST, "Zero Trust Architecture," NIST Special Publication 800-207, National Institute of Standards and Technology, Gaithersburg, MD, 2020. doi:10.6028/NIST.SP.800-207

[7] C. Paul et al., "Securing RPA Bots in Multi-Tenant Environments," ResearchGate, 2024. [Online]. Available: https://www.researchgate.net/publication/385593122

[8] M. Makinde et al., "Access Control Policies for Multi-Tenant SoC Environments," ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/394820040