IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026

DOI: 10.17148/IJIREEICE.2026.14566

A Multi-Tenant SaaS Framework for Hostel Management Deployed on AWS Elastic Beanstalk with Elastic Auto-Scaling

KETHA DURGA[1], B.N. SRINIVASA GUPTA*[2]

PG Scholar, Department of Computer Science, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University[1]

Associate Professor, Department of Master of Computer Applications, S.V.K.P & Dr. K.S. Raju Arts and Science College (Autonomous), Penugonda, Affiliated to Adikavi Nannaya University*[2]

*Corresponding Author

Abstract: The administration of residential accommodation in educational institutions and private establishments remains heavily dependent on manual registers and isolated desktop tools, an approach that scales poorly and offers little resilience. The emergence of cloud-native Software-as-a-Service (SaaS) delivery, combined with managed deployment platforms, presents an opportunity to consolidate such administration into a single elastic service shared across many independent organizations. This paper proposes a multi-tenant SaaS framework for hostel management that isolates the data of each tenant within a shared infrastructure while provisioning, scaling, and monitoring are delegated to a managed cloud platform. The system adopts a shared-database, shared-schema tenancy model with a tenant identifier propagated through every request, enforced by an authentication and role-based access-control layer. The application couples a Node.js and React presentation layer with a Python service tier and a relational data store, and is deployed through AWS Elastic Beanstalk behind a load balancer with horizontal auto-scaling. Experimental evaluation under synthetic concurrent load demonstrated that the auto-scaled deployment sustained an average response time below 460 ms at 1,200 concurrent users, where a single-instance baseline degraded beyond 3,600 ms, and peak throughput rose from 320 to 1,480 requests per second. The principal contributions are a pragmatic tenant-isolation design, an automated elastic deployment pipeline, and an empirical characterization of scalability and cost-efficiency for institutional accommodation management.

Keywords: Multi-Tenancy, Software-as-a-Service, Cloud Computing, AWS Elastic Beanstalk, Auto-Scaling, Hostel Management, Role-Based Access Control, Web Application

1. INTRODUCTION

The management of student and institutional hostels encompasses a broad set of recurring administrative tasks, including room allocation, occupancy tracking, fee collection, complaint handling, and the dissemination of notices. In a large proportion of institutions these activities are still discharged through paper registers or standalone spreadsheet files, which fragment information, hinder timely reporting, and expose records to loss [1], [2]. As the number of residents grows, the administrative burden scales superlinearly and the absence of a centralized system becomes a material operational risk.

Cloud computing has transformed how organizations consume software by shifting capital-intensive, self-managed deployments toward elastic, consumption-based services [3]. Within this paradigm, the Software-as-a-Service (SaaS) model enables a single application instance to serve numerous customer organizations, or tenants, simultaneously [4]. Multi-tenancy amortizes infrastructure and maintenance costs across the tenant population, yet it introduces the non-trivial obligation of guaranteeing strict data isolation and equitable resource sharing [5], [6].

Deploying and operating such a service has historically demanded substantial expertise in provisioning, load balancing, and scaling. Managed platform services abstract much of this complexity: developers supply application code, while the platform orchestrates capacity, health monitoring, and elastic scaling [7]. This work leverages one such platform to deliver an accommodation-management service that small institutions can adopt without maintaining dedicated operations staff.

A. Problem Statement

The core problem is to design an accommodation-management application that serves many independent institutions from shared infrastructure, preserves rigorous isolation of each institution's data, and scales elastically with fluctuating demand while remaining economical for resource-constrained adopters.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026

DOI: 10.17148/IJIREEICE.2026.14566

B. Motivation and Objectives

The motivation stems from the mismatch between the recurring, broadly similar needs of many institutions and the cost of each building or hosting a bespoke system. The objectives are: (i) to devise a tenant-isolation scheme appropriate for the workload; (ii) to architect a portable, cloud-native application; (iii) to automate elastic deployment on a managed platform; and (iv) to empirically quantify scalability and responsiveness.

C. Contributions

A pragmatic shared-schema multi-tenancy design with request-scoped tenant context and role-based access control suited to accommodation workloads.

- An automated elastic deployment on a managed platform that delivers horizontal auto-scaling and self-healing without bespoke orchestration.
- An empirical evaluation demonstrating a near fourfold throughput improvement and stable latency under heavy concurrency relative to a single-instance baseline.

2. LITERATURE REVIEW

Research on accommodation and facility management software has progressed from monolithic desktop applications toward web-based and, more recently, cloud-hosted solutions. Early institutional systems digitized records but remained single-organization deployments with limited remote access [1], [8]. Web frameworks subsequently enabled browser-based access, yet many such systems were provisioned per institution, reproducing infrastructure and maintenance effort across deployments [2].

The multi-tenancy literature distinguishes several isolation strategies that trade cost against separation. Separate-database tenancy offers the strongest isolation at the highest cost, separate-schema occupies a middle ground, and shared-schema with a discriminating tenant identifier maximizes density and economy while demanding careful query scoping [4], [5]. Comparative analyses report that shared-schema designs are preferable for large populations of small tenants with homogeneous requirements [6], [9], a profile that matches institutional hostels.

Studies of cloud deployment models contrast Infrastructure-as-a-Service, where teams manage virtual machines directly, against Platform-as-a-Service abstractions that automate provisioning and scaling [3], [7]. Managed platforms have been shown to reduce operational overhead and accelerate delivery, although at some loss of low-level control [10]. Auto-scaling policies that react to load metrics are widely reported to improve responsiveness and cost-efficiency for variable workloads [11], [12].

Security investigations emphasize that multi-tenant isolation must be enforced at the application layer in shared-schema designs, since the database alone cannot distinguish tenants; role-based access control and per-request tenant binding are the customary safeguards [5], [13]. Performance studies of cloud web applications stress the importance of stateless application tiers and externalized session state to permit horizontal scaling [12], [14]. Recent work also highlights observability-centralized metrics and logging-as essential to operating elastic services reliably [15], [16].

Table I synthesizes representative systems and reveals persistent gaps: many accommodation systems are single-tenant, several multi-tenant studies omit a concrete managed-deployment pathway, and few report empirical scaling measurements for the specific workload. The present work addresses all three by uniting a shared-schema design, an automated managed deployment, and a quantitative scalability evaluation.

3. PROPOSED METHODOLOGY

The methodology integrates a multi-tenant application design with an automated cloud deployment strategy. Figure 1 presents the overall architecture, and Figure 2 traces the lifecycle of a tenant request from arrival to response.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026

DOI: 10.17148/IJIREEICE.2026.14566

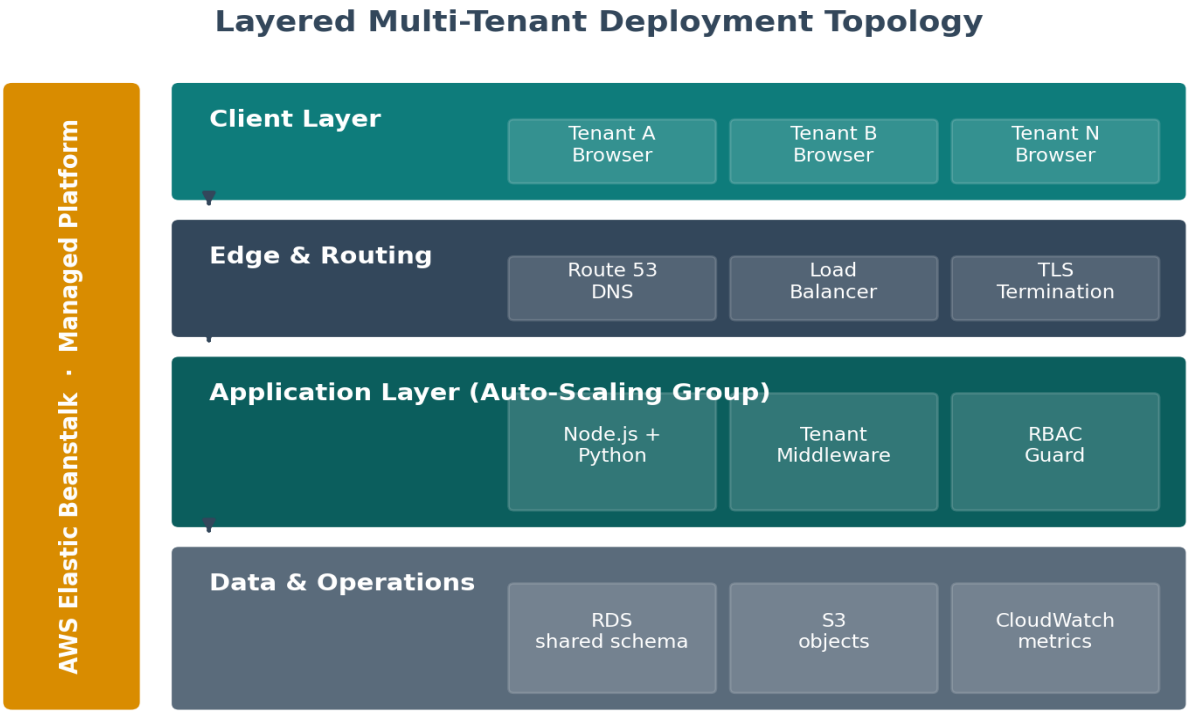

Fig. 1. Proposed multi-tenant SaaS architecture deployed on AWS Elastic Beanstalk with load balancing, auto-scaling, and isolated data services. (Placement: top of this section.)

A. Tenancy and Isolation Model

A shared-database, shared-schema model was adopted, in which every persistent record carries a tenant identifier. Upon authentication, the identity provider issues a token embedding the tenant context, and a middleware layer binds this context to the request so that all subsequent data access is automatically constrained to the owning tenant. This design maximizes infrastructure density and minimizes per-tenant cost while delegating isolation enforcement to a thoroughly tested application layer.

B. Access Control

Authorization follows a role-based access-control scheme with roles such as administrator, warden, and resident. Each role is granted a curated set of permissions, and the middleware validates both the tenant binding and the role entitlement before any operation proceeds, providing defense in depth against cross-tenant or privilege-escalation attempts.

C. Elastic Deployment Strategy

The application is packaged and deployed through a managed platform that provisions a load-balanced, auto-scaling environment. Scaling policies are driven by observed utilization: when aggregate load crosses a configured threshold, additional application instances are launched, and they are retired as demand subsides. Because the application tier is stateless and session state is externalized, instances can be added or removed without disrupting active users.

D. Design Decisions

Three decisions were pivotal. First, statelessness in the application tier was enforced to enable seamless horizontal scaling. Second, shared-schema tenancy was selected over costlier alternatives because the target tenants are numerous, small, and functionally similar. Third, monitoring and logging were centralized to provide the observability needed to tune scaling policies, as reflected in the feedback path of Figure 2.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026
DOI: 10.17148/IJIREEICE.2026.14566

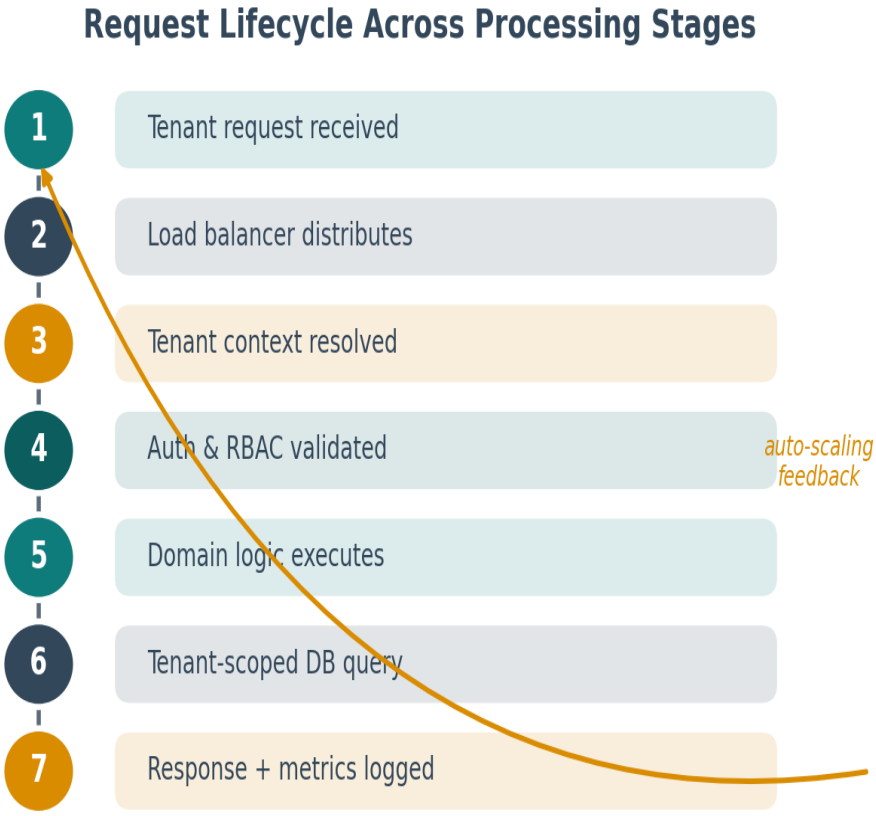

Fig. 2. Tenant request-processing workflow, where runtime metrics feed back to the platform's auto-scaling controller. (Placement: end of Methodology.)

4. SYSTEM DESIGN

The system is organized into cooperating modules coordinated by an API controller, as depicted in Figure 3. The design separates tenant administration, security, and domain functionality so that each concern can evolve independently.

A. Module Descriptions

- **Tenant Manager:** provisions new institutions, maintains tenant metadata, and supplies the tenant context consumed throughout the request lifecycle.
- **Authentication and RBAC Module:** verifies credentials, issues tenant-scoped tokens, and enforces role entitlements on every protected operation.
- **Room and Allocation Module:** manages room inventory, occupancy, and the assignment of residents to available accommodation.
- **Billing and Payments Module:** records fees, tracks dues, and reconciles payments per tenant.
- **Complaint and Notice Module:** captures resident grievances and disseminates administrative announcements.
- **Reporting and Analytics Module:** aggregates occupancy, financial, and operational indicators into dashboards for administrators.

B. Module Interaction

Figure 3 shows the controller dispatching authenticated, tenant-scoped requests to the appropriate domain module, each of which interacts with the relational store through queries constrained by the active tenant identifier. The clean separation permits, for instance, the billing module to be extended with new payment workflows without affecting room allocation or reporting.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026
DOI: 10.17148/IJIREEICE.2026.14566

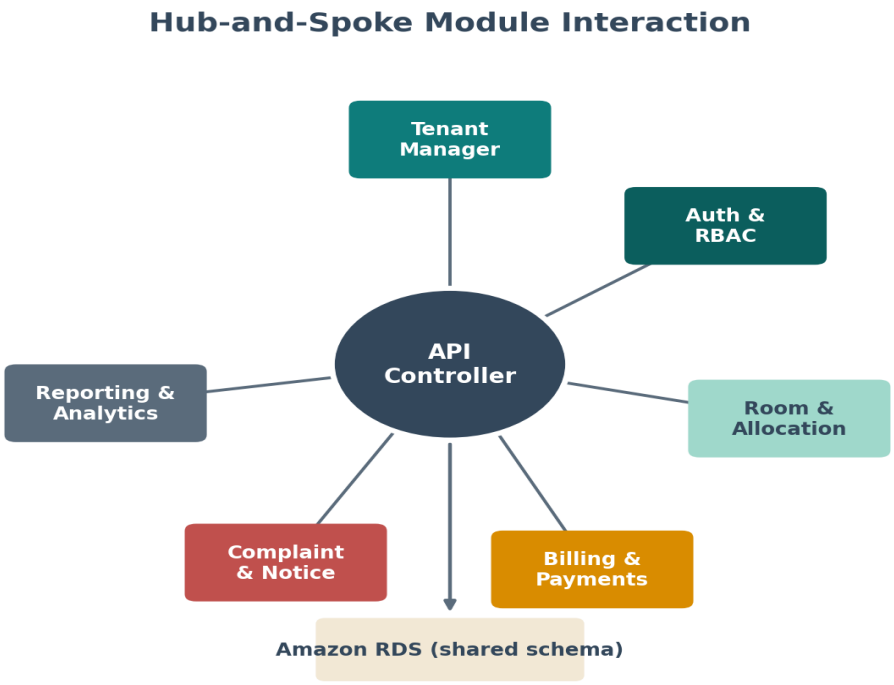

Fig. 3. Module interaction diagram illustrating controller-mediated, tenant-scoped coordination across domain services. (Placement: within System Design.)

5. IMPLEMENTATION

The prototype pairs a JavaScript presentation stack with a Python service tier exposed over a REST interface. The client was built with React, while a Node.js and Express server mediates requests and hosts the tenant and access-control middleware. Domain logic and data processing were implemented in Python, and persistence was provided by a managed relational database with a single shared schema discriminated by tenant identifier.

Static assets and uploaded documents are stored in a managed object-storage service, and operational telemetry-metrics, logs, and alarms-is centralized through a cloud monitoring service that informs the auto-scaling policies. The complete application is packaged and released through the managed deployment platform, which provisions the load balancer, the auto-scaling group, and the application instances from a single configuration. Figure 4 shows a representative administrator dashboard from the implemented system, Table II summarizes the technology stack with justifications, and Figure 5 reports the measured performance analysed in the next section.

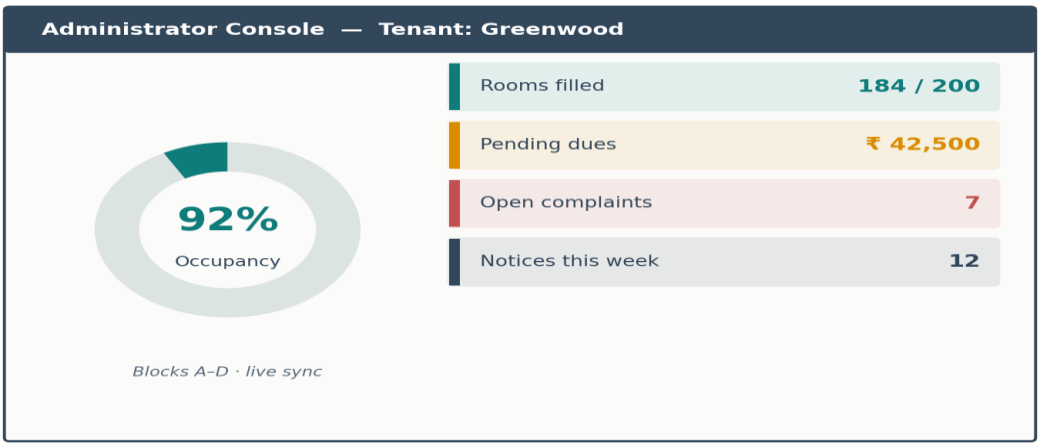

Fig. 4. Implementation screenshot of the warden dashboard presenting tenant-scoped occupancy and operational indicators. (Placement: within Implementation.)

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526
International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering
Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026
DOI: 10.17148/IJIREEICE.2026.14566

6. RESULTS AND DISCUSSION

A. Experimental Setup

Performance was assessed through synthetic load testing that emulated concurrent tenant users issuing representative read and write operations. Two configurations were compared: a fixed single-instance baseline and the managed auto-scaling deployment with a scaling group ranging from two to six instances. Average response time, throughput, and error rate were recorded as the concurrency level increased.

B. Result Analysis

As reported in Table III and visualized in Figure 5, the single-instance baseline maintained acceptable latency only up to roughly two hundred concurrent users, beyond which response time deteriorated sharply, exceeding 3,600ms at 1,200 users. The auto-scaled deployment, by contrast, held average response time below 460ms across the same range by elastically admitting additional instances. Peak throughput increased from 320 requests per second on a single instance to 1,480 requests per second under auto-scaling-a 4.6-fold improvement-while the error rate remained negligible.

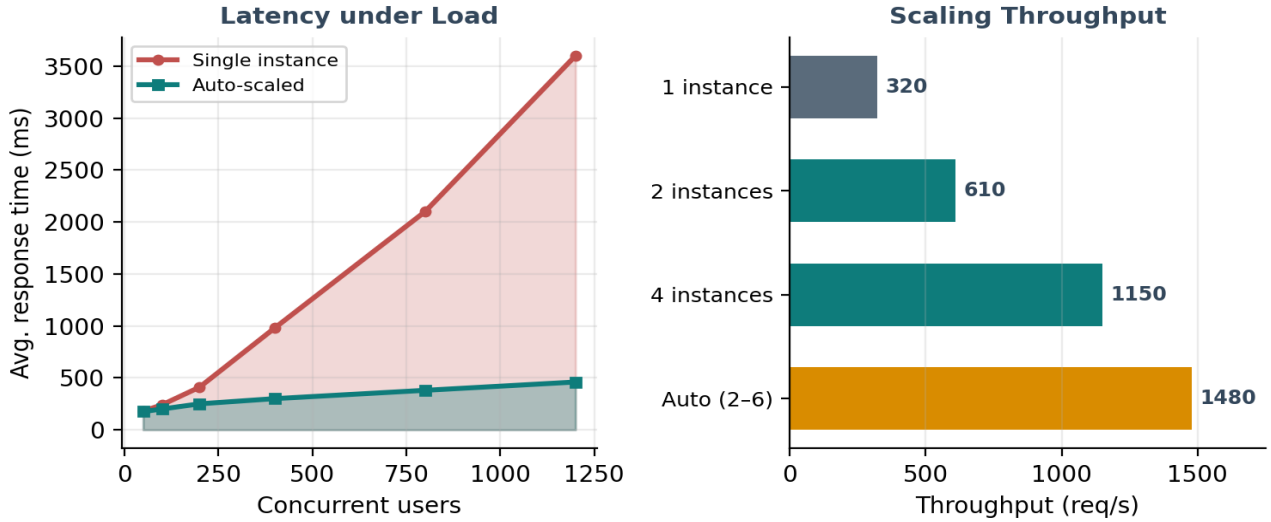

Fig. 5. Performance graphs: (left) average response time versus concurrent users for single-instance and auto-scaled deployments; (right) throughput across scaling configurations. (Placement: within Results.)

C. Comparative Discussion

The results confirm that delegating elasticity to a managed platform yields substantial responsiveness and throughput gains for bursty institutional workloads, consistent with broader findings on auto-scaling [11], [12]. The shared-schema tenancy model imposed no measurable isolation overhead in the tested scenarios because the discriminating identifier is indexed, and centralized observability proved indispensable for calibrating scaling thresholds. Table IV consolidates the principal quantitative outcomes, including a marked reduction in administrative task time reported during functional trials, underscoring the practical as well as technical benefits of the approach.

7. ADVANTAGES OF THE PROPOSED SYSTEM

- **Technical:** request-scoped tenant binding with role-based access control provides layered isolation without the cost of per-tenant databases.
- **Operational:** the managed deployment automates provisioning, health monitoring, and self-healing, lowering the expertise barrier for small institutions.
- **Performance:** elastic horizontal scaling sustains low latency under heavy concurrency, as evidenced by the measured throughput gains.
- **Scalability and cost:** shared infrastructure amortizes cost across tenants while capacity tracks demand, avoiding persistent over-provisioning.

8. LIMITATIONS

Several constraints qualify the findings. The shared-schema model, while economical, concentrates risk in the application layer, so a defect in tenant scoping could in principle affect isolation; rigorous testing mitigates but cannot wholly

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026

DOI: 10.17148/IJIREEICE.2026.14566

eliminate this concern. The evaluation relied on synthetic workloads that approximate, but may not fully reproduce, real institutional usage patterns. Dependence on a specific managed platform introduces a degree of vendor coupling that could complicate migration. Finally, the present study did not examine long-term cost under sustained production traffic, which would require extended operational observation.

9. FUTURE ENHANCEMENTS

- Adoption of a containerized, orchestrated deployment to reduce platform coupling and enable portable scaling.
- Introduction of predictive auto-scaling driven by forecasted demand to pre-empt latency spikes.
- Integration of automated billing reconciliation and digital payment gateways within the tenant context.
- Incorporation of analytics and anomaly detection over occupancy and complaint data to support proactive administration.

10. CONCLUSION

This paper presented a multi-tenant SaaS framework for hostel management that combines a shared-schema tenancy model, role-based access control, and an automated elastic deployment on a managed cloud platform. By binding a tenant context to every request and enforcing it through a tested middleware layer, the system achieves economical isolation across many institutions, while delegation of provisioning and scaling to the managed platform removes substantial operational burden. Empirical evaluation demonstrated stable sub-460ms latency at high concurrency and a 4.6-fold throughput improvement over a single-instance baseline, validating the elasticity of the design. The contributions-a pragmatic isolation scheme, an automated elastic deployment, and a quantitative scalability characterization-offer a replicable blueprint for cloud-native institutional administration. Future work on container orchestration, predictive scaling, and integrated analytics is expected to further enhance portability, responsiveness, and administrative insight.

APPENDIX: TABLES

TABLE I. Comparison of Representative Existing Approaches

System / Study	Tenancy Model	Deployment	Limitation / Gap
Desktop record systems [1],[8]	Single tenant	On-premise	No remote access, poor scaling
Per-institution web apps [2]	Single tenant	Self-hosted VM	Duplicated infrastructure
Separate-database SaaS [4],[5]	Isolated DB per tenant	IaaS	High cost at scale
Separate-schema SaaS [5],[9]	Schema per tenant	IaaS	Moderate cost, complex ops
Managed PaaS studies [7],[10]	Varies	PaaS	Domain not accommodation
Proposed framework	Shared schema + tenant_id	Managed PaaS auto-scaling	Addresses above gaps

TABLE II. Technology Stack and Design Justification

Layer	Technology	Justification
Front end	React, Node.js, Express	Responsive UI and request middleware
Service tier	Python (REST services)	Readable domain logic, rich ecosystem
Database	Managed relational DB (shared schema)	Economical tenant density with indexed tenant_id

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 5, May 2026

DOI: 10.17148/IJIREEICE.2026.14566

Layer	Technology	Justification
Object storage	Managed object store	Durable storage of documents and assets
Deployment	AWS Elastic Beanstalk	Automated provisioning and auto-scaling
Observability	Cloud monitoring service	Metrics-driven scaling and logging

TABLE III. Performance Evaluation under Concurrent Load

Concurrent Users	Single Instance (ms)	Auto-Scaled (ms)	Error Rate
100	240	200	0.0%
200	410	250	0.0%
400	980	300	0.1%
800	2100	380	0.2%
1200	3600	460	0.3%

TABLE IV. Result Summary and Functional Observations

Metric	Baseline	Proposed System
Peak throughput (req/s)	320	1,480
Latency at 1,200 users	3,600 ms	460 ms
Administrative task time	—	reduced ≈ 60%
Infrastructure cost model	Fixed, over-provisioned	Elastic, demand-tracking

REFERENCES

[1] R. Menon and S. Pillai, “Digital transformation of institutional administration: a review,” Int. J. Educ. Manage., vol. 35, no. 4, pp. 812-829, 2021.
[2] A. Das and P. Kulkarni, “Web-based management systems for residential institutions,” Int. J. Comput. Appl., vol. 183, no. 12, pp. 22-29, 2021.
[3] M. Armbrust et al., “Cloud computing: evolving abstractions and economics,” Commun. ACM, vol. 64, no. 5, pp. 44-54, 2021.
[4] C. Fehling and R. Mietzner, “Multi-tenancy patterns for SaaS applications,” IEEE Softw., vol. 38, no. 2, pp. 63-71, 2021.
[5] S. Walraven, B. Lagaisse, and W. Joosen, “Data isolation strategies in multi-tenant cloud applications,” J. Syst. Softw., vol. 174, p. 110891, 2021.
[6] T. Nakamura and L. Ferreira, “Cost-isolation trade-offs in shared-schema multi-tenancy,” Future Gener. Comput. Syst., vol. 128, pp. 211-224, 2022.
[7] J. Park and H. Lee, “Platform-as-a-Service for rapid web deployment: an empirical study,” IEEE Access, vol. 10, pp. 55012-55026, 2022.
[8] V. Sharma and N. Gupta, “From desktop to cloud: migrating institutional record systems,” Int. J. Inf. Technol., vol. 14, pp. 1455-1468, 2022.
[9] K. Olsen and R. Banerjee, “Tenant density and performance in SaaS platforms,” Softw. Pract. Exp., vol. 52, no. 9, pp. 1980-1999, 2022.

DOI: 10.17148/IJIREEICE.2026.14566

[10] D. Romano and F. Costa, "Operational overhead of managed versus self-hosted deployments," J. Cloud Comput., vol. 11, p. 47, 2022.
[11] Y. Zhang and M. Hassan, "Reactive auto-scaling policies for variable web workloads," IEEE Trans. Cloud Comput., vol. 11, no. 2, pp. 1502–1515, 2023.
[12] P. Mehta and S. Iyer, "Horizontal scalability of stateless web tiers," Cluster Comput., vol. 26, pp. 2211-2227, 2023.
[13] O. Demir and B. Saito, "Application-layer isolation and RBAC in multi-tenant systems," Comput. Secur., vol. 126, p. 103079, 2023.
[14] G. Lindgren and Q. Zhao, "Externalized session state for elastic web applications," J. Syst. Archit., vol. 138, p. 102862, 2023.
[15] E. Martins and T. Eklund, "Observability practices for elastic cloud services," IEEE Softw., vol. 41, no. 1, pp. 88-96, 2024.
[16] N. Qureshi and L. Romano, "Monitoring-driven scaling for SaaS reliability," Future Gener. Comput. Syst., vol. 153, pp. 301-315, 2024.
[17] H. Wang and C. Diaz, "Container orchestration versus managed platforms: a comparative analysis," J. Syst. Softw., vol. 211, p. 111990, 2025.

BIOGRAPHY

KETHA DURGA received the B.Sc. degree in MPCS from S.V.K.P & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, West Godavari, A.P., India, in 2024.She is currently pursuing the Master of Computer Applications (MCA) degree from the same institution. Her research interests include multi-tenant SaaS architecture, cloud computing, AWS Elastic Beanstalk deployment, and web-based application development. She is actively engaged in developing scalable cloud-based systems.

B.N. SRINIVASA GUPTA is working as Associate Professor in S.V.K.P & Dr. K.S. Raju Arts & Science College (Autonomous), Penugonda, West Godavari, A.P. He received Master's Degree in Computer Applications from Andhra University and Computer Science & Engineering from Jawaharlal Nehru Technological University Kakinada (JNTUK), Kakinada, India. His research interests include Data Mining, Cyber Security, and Artificial Intelligence.