AI-Enabled Architecture for Large-Scale Multi-Tenant Systems

Rajesh Cherukuri[1*], Siva Karthik Parimi[2]

[1] Senior Software Engineer, PayPal, Austin, TX, UNITED STATES
[2] Senior Software Engineer, PayPal, Austin, TX, UNITED STATES

*Corresponding Author Email: rajeshcherukuri@icloud.com

ABSTRACT

A Software-as-a-Service provision delivers pre-configured solutions for its clients. Tenants with varying quality requirements are assigned to distinct dedicated instances. The issues associated with producing inconsistent quality responses from a single instance are not explicitly addressed in current design methodologies. The absence of standardized strategies and design protocols complicates an architect's ability to incorporate multi-tenant design choices during the initial phase. This work emphasizes several domain-independent architectural considerations for managing multiple heterogeneous tenants on a shared application instance. We identify essential quality requirements pertinent to the multi-tenant scenario, associated tactics, metrics, and assess their influence on other software product quality attributes.

Keywords: AI-Enabled Architecture; Multi-Tenant Systems; Platform Governance; Intelligent Optimization; Tenant Isolation; Enterprise Cloud Platforms

Introduction

A multi-tenant application enables multiple tenants with diverse requirements to share a single application and database instance. Even these tenants experience tailored functional and quality responses as delivered by a dedicated instance. It resembles a multi-processing scenario at the operating system level, where each process operates independently and receives varying responses from the underlying operating system based on its priority, scheduling, and threading preferences. In a comparable manner, tenants represent processes, while multi-tenant applications correspond to an operating system. A multi-tenant application produces tailored responses according to the needs of each tenant. Software architectural tactics are employed to attain specified quality responses from the software application. These strategies operate at various levels. System-level tactics necessitate system utilities for their implementation. These tactics cannot be implemented at the application or database level [1]. Examples of such tactics include the regulation of CPU cycles and memory utilization. Conversely, application-level strategies, including hot redundancy of microservices and runtime workflow orchestration, are managed at the application tier. These microservices and workflow components lack any control mechanisms at the system level. This study identifies issues associated with the implementation of operating system-like behavior for tenants within multi-tenant applications. Our primary emphasis is on the tenant management dimensions of elasticity, observability, and separability [2].

The principal contributions delineated herein are:

• Identification of specific, measurable architectural quality attributes for multi-tenant systems and associated tactics for their realization, in addition to the quality attributes outlined in the ISO/IEC 25010 software product quality model [3].

• A pragmatic method for the objective assessment of multi-tenant applications.

• Preliminary assessment of the influence on overall software product quality characteristics.

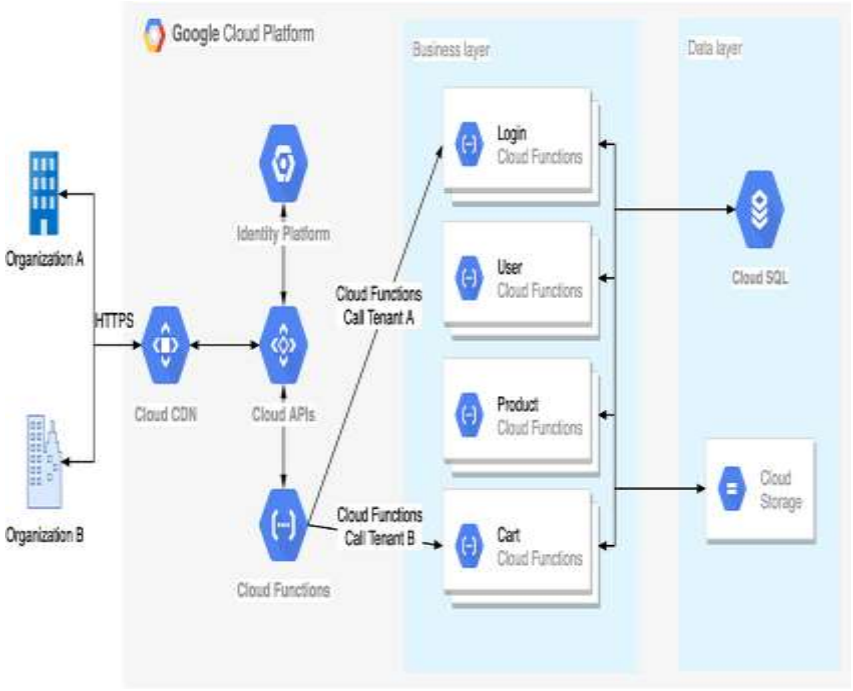

Figure 1. Expanding the general software quality model to encompass cloud SaaS and multi-tenant systems.

Methodology

We succinctly elucidate our methodology herein. We formulated the definition of multi-tenant SaaS applications based on the definitions of multi-tenant and SaaS applications. Subsequently, based on this definition, we delineated several generic business requirements and refined them into specific quality issues. These quality concerns are categorized into three overarching quality attributes [4].

Figure 1 elucidates our methodology for formulating the quality model for multi-tenant SaaS applications. We examine the definitions of Multitenant application, tenant, and Software-as-a-Service to identify essential requirements pertinent to multi-tenant SaaS applications. A multi-tenant application enables customers (tenants) to utilize shared hardware resources through a singular application and database instance, while permitting customization to meet their specific requirements as though operating in a dedicated environment.

A tenant is the organizational entity that leases a multi-tenant SaaS solution. A tenant usually aggregates several users, who represent the stakeholders within the organization. Software-as-a-Service (SaaS) is a software licensing and delivery model wherein software is licensed on a subscription basis and hosted centrally [5].

Utilize identical hardware resources: Service providers must monitor each tenant to ensure compliance and prevent interference with other tenants. Monitoring the tenants also assists the service provider in coordinating tenants on a singular shared instance of the application. Conversely, it can assist developers in debugging and maintenance activities.

Configure the application to accommodate their requirements: It is preferable to enable tenants to independently customize their services. Given the potential multitude of tenants on a collection of MTSA instances, it may be unfeasible for a service provider to manage service configurability for each tenant. The service's efficiency is enhanced as tenants can

independently reconfigure their personalized services without the service provider's intervention.

A dedicated environment: The tenant must operate independently without disrupting its co-hosted tenants. Interference may arise from the dynamic nature of requirements and the fluctuating workload of tenants [6].

Rental durations vary, with some tenants being transient and others remaining long-term. In the execution of an MTSA instance, enduring tenants may be co-located with transient tenants. A MTSA must manage each tenant autonomously and eliminate the tenant's residuals post-termination for optimal resource management.

Aggregates multiple users: Frequently, the workload of a tenant may be unpredictable in advance. Tenant workloads may fluctuate over time due to changing business objectives and requirements. In this scenario, the scheduling of co-hosted tenants must be executed appropriately, considering that the quality requirements of all tenants may fluctuate dynamically. Neglecting the uncertainty in tenant workload behavior may lead to the tenant becoming incompatible for co-hosting with other tenants.

Subscription model: Tenants should only pay for the resources they have utilized. Consequently, it is essential to assess the resource consumption and operational expenses for each tenant individually.

We classify these quality requirements according to the support needed from the underlying multi-tenant system into three principal quality attributes: Tenant Elasticity, Tenant Separability, and Tenant Observability. In accordance with these requirements, the mapping of various quality attributes, QAs, and illustrative scenarios is presented in Table I. The subsequent section provides a detailed explanation of each QA [7].

TABLE 1: BUSINESS REQUIREMENTS AND QUALITY REQUIREMENT SCENARIO

Business requirement	Refinement	Quality Concern	Quality Attribute	Quality requirement scenario
Allow a tenant to self-configure	A tenant wants to increase availability at runtime A tenant wants to add new payment options	Variable Workload Service Configurability	Elasticity Elasticity	A tenant should be able to reconfigure its availability from “98%” to “99.99%”. A tenant should be able to add new payment gateways at runtime without service provider’s intervention.
Co-host tenants in logically separated manner	Modification in the configuration for a tenant should not impact other tenants A tenant’s short-term goal is completed, and it wants to shut down its service	Independence Non-identical lifespan	Separability Separability	A tenant should be able to reconfigure its availability from “98%” to “99.99%” without impacting the availability of other tenants. A tenant should be able to exit from the multi-tenant instance at any time without service provider’s intervention.
System level measures should be mapped to tenant level measures	The service provider wants to monitor each tenant activity for generating invoices The service provider wants to track the footprint of each tenant for uninterrupted service	Metering Monitoring	Observability Observability	The service provider wants to measure CPU, memory, storage and network consumption for each tenant. The service provider watch each tenant during their dynamic reconfiguration to ensure if the new set of configurations is not creating any interruption for other co-hosted tenants.

Quality Attributes of Multi-Tenant Systems

This section delineates and examines the three quality attributes pertinent to MTSA. These QAs elucidate the quality requirements of various stakeholders and assist in the architectural design of MTSAs, ensuring efficient tenant management and reduced maintenance overhead. Each QA description is organized according to the following structure [8]:

• Definition of Quality Assurance

A Quality Attribute Scenario (QAS) is a standardized method for articulating requirements pertaining to quality attributes [3]. Table II presents the QAS associated with each QA.

TABLE 2: QUALITY ATTRIBUTE SCENARIOS FOR MULTI-TENANT QAS

Quality Attribute	Source	Stimulus	Artifact	Environment	Response	Response Measure
Tenant Elasticity: Variable Workload	Tenants short-term variability, external to the system	Unanticipated tenant load	Process, Processor, Communication	Normal Operation	Adapt to new workload requirements by changing the system configuration	Adaptation time, Tenant interference
Tenant Elasticity: Service Configurability	Tenants planned variability, external to the system	Changing business requirements	Process, Processor, Communication	Normal Operation	Tenant modify configuration	Easiness for tenants, Tenant flexibility range, Tenant interference
Tenant Separability: Independence	Tenants	Changing tenant configurations	Process, Processor, Communication	Normal Operation	Migrate the tenant to a different multi-tenant instance or dedicated instance	Time to migrate the tenant, Tenant interference
Tenant Separability: Non-identical lifespan	Tenants	Termination Of contract, Tenant's business requirements	MTSA	Normal Operation	Terminate a tenant	Effort to terminate and cleanup residuals, tenant interference
Tenant Observability: Metering	Internal to the system	Invoicing tenants	Process, Processor, Communication, Storage	Normal Operation	Calculate resource usage for each tenant	Number of exposed parameters with tenant level measurements, Accuracy of tenant level measures
Tenant Observability: Monitoring	Internal to the system	Tracing activities of each tenant	Process, Processor, Communication, Storage	Normal Operation, Degraded Operation, Debugging	Read tenant level parameters	Number of exposed parameters with tenant level measurements, Accuracy of tenant level measures

Tactics are established methods for implementing quality attributes within an application [3].

• Metrics and measurements for assessing quality attributes.

• Instances of current MTSAs that exemplify the QA.

A. Tenant Elasticity

Tenant-level elasticity accommodates the dynamic behavior of tenants. The needs of tenants may evolve over time. The tenant elasticity quality attribute pertains to the application's capability to enable its tenants to alter their requirements during runtime. It is conceptually analogous to the hypervisor's capacity to alter virtual machine configurations during runtime. The application instance's state is represented by a synthesis of configuration files, data files, CPU, cache, and database states. The alteration of a tenant's requirements dynamically allocates and configures resources according to the new specifications and modifies the application's state.

Definition: The capacity of an MTSA to adjust to the evolving demands of tenants. It is a meta quality assurance, assessed in relation to the variability of other quality assurances provided by the application. Quality Requirement Scenario: An illustration of a quality requirement scenario for tenant elasticity is as follows: Tenant A seeks to transition from

low performance (response time ≤ 10 seconds) to high performance (response time ≤ 5 seconds). The strategies for implementing tenant elasticity closely resemble those for runtime variability [4]. We outline several of these strategies here:

Deferring the binding of configuration variables enables dynamic modification of tenant-specific configurations, providing flexibility to tenants. [7]

Tenant-aware dynamic resource allocation can fulfill their evolving needs.

Dynamic architecture-based components provide a means to attain elasticity by constructing components that accommodate runtime variability, thereby allowing modifications to the extent of quality attributes [3-5].

• Runtime workflow orchestration, informed by contextual factors, meets the dynamic requirements of the tenants.

Metric & Measurement: Tenant elasticity pertains to the variability of all other quality attributes (availability, performance, security, etc.) in accommodating the dynamic requirements of tenants. Consequently, it intuitively seems that increased variability and options are advantageous. According to the principle of Occam's razor, we define the elasticity of a QA as the aggregate number of available options for it.

A MTSA may possess multiple QAs, each accompanied by a corresponding list of options. The elasticity of each QA is measured by the number of options available. Nevertheless, it is imperative to quantify the elasticity of MTSA. This necessitates a synthesis of the elasticity scores for each QA. A straightforward weighted sum of all QAs may be employed. For varying ranges of QAs, a normalized range should be employed to compute the application's elasticity index. For instance, two multi-tenant applications with identical functionality may accommodate varying ranges of quality assurances.

QA2: Availability alternatives: {99%, 99.9%}

In this instance, application A exhibits greater elasticity concerning availability, while application B demonstrates enhanced elasticity for throughput requirements.

Tactics: Tenant-aware metering and logging are two methods to enhance observability in an MTSA. Logs that are cognizant of tenants can be examined to deduce metering information for each tenant.

Metric & Measurement: The observability of a multi-tenant application for an ASP can be quantified as the ratio of the number of relevant quality measures provided at the tenant level to the total number of pertinent quality measures for the ASP.

It is important to recognize that evaluating the elasticity of a QA based on the number of options constitutes a rather simplistic model. We have not considered various factors, such as the different potential values available for a specific tenant based on a given value of QA.

Examples include several multi-tenant systems that incorporate tenant elasticity. An exemplary instance that offers meticulous control at the tenant level is the IBM multi-tenant JVM [6]. We can evaluate the tenant elasticity of the IBM multi-tenant JVM in comparison to the open-source JVM. The IBM JVM allows for the regulation of processor time, heap

size, thread count, file I/O, socket I/O, and other parameters according to tenant requirements, thereby providing enhanced tenant elasticity.

B. Tenant Surveillance

A tradeoff exists between the granularity of metadata for application monitoring and the associated overhead in monitoring [8]. Nevertheless, current measurements operate at both the process and thread levels. Tenant-level monitoring assesses diverse system and application parameters for each tenant to evaluate the quality of responses. Tenant observability of an application facilitates the assessment of quality attributes at the tenant level, offering a more refined granularity than the process level. As tenants utilize a shared application instance and do not correspond to distinct threads or processes, monitoring metrics for each tenant becomes challenging.

In an MTSA, it is essential to gather tenant-level metrics, including response time, resource utilization, and disk I/O frequency. A multi-tenant application featuring tenant observability enables Application Service Providers (ASPs) to monitor all tenants with greater precision and implements a pay-as-you-go model. It also aids in the root cause analysis of defects and the identification of unforeseen tenant behavior. It enables tenants to ascertain their compliance with quality standards by evaluating software product quality metrics.

including resource utilization, temporal behavior, and efficiency adherence.

Definition: The capacity of an MTSA to reveal system and machine-level metadata at the granularity of individual tenants.

Quality Requirement Scenario: An illustration of a quality requirement scenario for tenant observability involves the MTSA service provider's desire to monitor the CPU utilization of a specific tenant [9].

O: Observability index for the multi-tenant application.

SQM: The aggregate count of software quality measures related to the ASP and facilitated by the multi-tenant application at the tenant level of granularity.

CQM: The aggregate of software quality metrics pertaining to the ASP.

Observability is defined to address the concerns of the ASP. ASP frequently necessitates a limited selection of parameters for measurement, contingent upon domain requirements. Monitoring all parameters is infeasible for an application without considerable overhead. An ASP can utilize the observability index to quantify the support provided by various multi-tenant applications for metering and monitoring purposes. The SQM and CQM parameters are contingent upon the requirements of the ASP, resulting in variability in the observability index of a multi-tenant application across different ASPs [10].

Zendesk [10] is a multi-tenant CRM-as-a-service that facilitates swift and straightforward interactions between businesses and their clients. The application facilitates the measurement of response time for service requests submitted by clients across all communication channels at the tenant level. The IBM multi-tenant JVM [6] provides monitoring capabilities at the tenant level to regulate the utilization of processor time, heap memory size, and other resources.

C. Tenant Independence

A service provider must manage tenants distinctly and autonomously. The situation resembles virtual machine administration in an Infrastructure-as-a-Service (IaaS) context. All challenges and issues associated with VM management are similarly present in tenant management. Nonetheless, tenant migration is more intricate owing to the elaborate design of the MTSA. Given that all tenants utilize a singular application instance, migrating a tenant to a different instance necessitates additional effort. Various degrees of separability may exist. A multi-tenant application may provide data layer separability through the implementation of a multi-tenant database schema [11]. At the application layer, a tenant may be distinct if a multi-tenant application permits the retrieval and modification of each tenant's workflow configuration and application-level parameters independently.

Separability is the metric that assesses the simplicity of eradicating a tenant's entire footprint from the application instance without affecting other tenants. Inseparability may also adversely affect other quality-of-service dimensions, including security, reliability, and resource efficiency.

Definition: The capacity of a multi-tenant application to segregate the data and state of an individual tenant.

Quality Requirement Scenario: A quality requirement scenario for tenant separability is that the application service provider intends to reutilize the resources utilized by a tenant following the termination of the service contract.

Strategies for achieving tenant separability should be implemented at every layer: data, application, and presentation. In the data layer, multi-tenant database schema design offers differing levels of separability among tenants, as examined in the context of relational databases. Non-relational databases like MongoDB offer various levels of tenant-specific configuration options, including the creation of distinct collections with unique indexes. At the application layer, disjoint reentrant code and tenant-specific code improve separability. Customized themes and widgets are currently being implemented in numerous systems at the user interface layer.

Metric & Measurement: The tenant separability of the application is defined as the separability at the level of distinct components. An application component is tenant-separable if it can recognize and eliminate a tenant-specific state. Consequently, a straightforward ratio of the tenant's separable components to the total components of the application serves as an effective metric for separability. Once more, we have employed a rudimentary representation of a system. One might also formulate a more intricate metric.

Oracle Business Intelligence Fusion Middleware offers various methods for tenant removal. It eliminates the tenant by eradicating tenant-specific directories. It also provides options to eliminate tenant identity from the identity management module and the tenant's data sources.

Verification of suggested metrics

Every quality attribute possesses corresponding metrics. The metrics must conform to the criteria established in the IEEE 1061 standard for a Software Quality Metrics Methodology

[4]. It outlines six criteria for validating a metric pertaining to software quality requirements: correlation, tracking, consistency, predictability, discriminative power, and reliability. We consider only the following three criteria out of six.

Correlation: The metric utilized to assess the proposed set of quality attributes must exhibit a robust linear relationship with the quality attributes and ensure consistency in quantification. Tenant observability is directly proportional to the quantity of software quality measures (SQM) relevant to the service provider.

Consistency: The varying metric values should align with the differing levels of quality attributes in a coherent sequence. If two metric values, Ma and Mb, satisfy the relation Ma < Mb, then the corresponding degrees of quality attributes QAa and QAb will exhibit the same order. QAa is less than QAb. The quality attribute with greater variability enhances the application's elasticity.

Discriminative power: The metric must effectively differentiate between varying degrees of quality attributes. For instance, the collection of metric values linked to a high quality of separability must be markedly greater (or lesser) than the metric value linked to a diminished degree of separability.

The remaining three criteria—tracking, predictability, and reliability—are not relevant for the initial analysis. Tracking evaluates the ability of a metric to monitor variations in the quality of a product or process throughout its life cycle. Predictability criteria pertain to accuracy. We do not regard it as the proposed metrics are in a preliminary stage and require further refinement. Reliability criteria guarantee that the metric has successfully undergone validity testing across a substantial number of applications.

Impact of Multi-Tenant Quality Assurance Systems

This section assesses the influence of the previously discussed tactics on overall software product quality and deduces the correlation with MTSA QAs. We consult the ISO 25010 software product quality model for the definitions of general quality attributes. Table III encapsulates our findings regarding the impact of multi-tenancy QAs on the integration of other general quality attributes within the system. The “+” symbol in the table signifies that the multi-tenant QA positively reinforces the overall QA. Conversely, the “-” symbols denote the adverse effect.

The influence of tenant elasticity allows for dynamic growth and contraction at runtime with minimal or no downtime, thereby improving tenant availability. The implementation of tactics such as defer binding and runtime resource allocation to enhance elasticity diminishes application performance. Moreover, deferred binding complicates system testing. It also heightens potential security threats by exposing runtime variables to the tenants.

The influence of tenant observability enables the identification of anomalous tenant behavior and the early detection of faults. It diminishes the time required for repairs and enhances availability. The monitoring process impedes system performance and adversely affects efficiency. Debugging and testing MTSA with tenant-level parameters is more straightforward. Observability may necessitate specific system-dependent tools to assess essential parameters. In these situations, observability adversely affects portability.

The impact of tenant separability: The separability quality attribute can occasionally reduce tenant availability due to the exclusive lock on shared critical resources during the separation process. Conversely, it facilitates the application's ability to execute garbage collection at the tenant level and eliminate the remnants of vacated tenants. It enables the individual tracking of each tenant and facilitates debugging and testing. Separability also aids in diminishing interference among tenants.

Associated Research

No work was identified that formally addresses quality concerns for multi-tenant SaaS applications. Authors in [6] introduced a multi-tenant data architecture within the framework of relational database systems. Oracle developed its container database.

Cloud Database (CDB) employs a multi-tenant architecture to minimize costs, facilitate performance optimization and maintenance, enable database consolidation, and support resource management [11]. Both works are confined to multi-tenancy at the database tier. The study in [7] examined multi-tenancy across various layers, including the kernel, database, application development, and query processing, within the framework of cloud application development. All of these works are domain-specific case studies on the implementation of multi-tenant applications. Authors in [1] examine how an erroneous decision can render multi-tenancy a costly endeavor rather than reducing maintenance overhead and expenses. Our research tackles this issue and delineates the architectural quality attributes of a multi-tenant application, irrespective of the domain, which can serve as a foundation for assessing application quality and informing suitable architectural design choices.

TABLE 3: RELATIONSHIP AMONG QAS. "+" SHOWS THAT ONE QA INCREASES THE DEGREE OF OTHER QA WHEREAS "-" CONVEYS THE TRADE-OFF

Quality Attributes	Availability	Performance	Testability	Security
Tenant Elasticity	+ (allows a tenant to modify itself without any downtime)	- (requires to bind variables at runtime)	- (late binding allows variables to have unpredictable values and make it hard to test)	- (exposure of runtime variables to tenants makes the application vulnerable to attacks)
Tenant Observability	+ (observability helps in early detection of faults and increase the availability)	- (overhead of resource consumption to measure metrics)	+ (allows to collect more metadata and make the testing task easier by locating bugs)	+ (allows to keep track of abnormal behavior of tenants)
Tenant Separability	- (separation process may lock some shared resource and bring down the availability for other tenants.) + (runtime failure in a tenant's logical space may not impact the normal operations of other co-hosted tenants.)	+ (may improve the performance of individual tenants by reducing the interference) - (overall efficiency may suffer due to resource usage overhead)	+ (increase the logical separation among tenants)	+ (reduce the security concerns by removing the unnecessary tenant residuals and decreasing the interference)

Conclusion and Future Research

This study delineates relevant quality attributes of an MTSA, specifically Tenant Elasticity, Observability, and Separability. These QAs are either absent or insignificant in a single-tenant application. Our preliminary examination of the current multi-tenant systems—Microsoft Dynamics 365, Oracle Business Intelligence, and IBM Multi-tenant JVM—

indicates that these characteristics are manifested to differing extents. Nevertheless, they appear to be inadequately addressed. Explicitly addressing these architectural concerns regarding effective tenant management is beneficial for designing and assessing the quality of a multi-tenant application, alongside the established software product quality attributes. This study also delineates several appropriate strategies for implementing these QAs. The existing collection of multi-tenant QAs identified in this study is incomplete. It exclusively concentrates on tenant management within multi-tenant environments. An extension of this research is to conduct a comprehensive quantitative analysis of current multi-tenant specific QAs.

References

[1] Tulli, S.K.C. (2022) Technologies that Support Pavement Management Decisions Through the Use of Artificial Intelligence. International Journal of Modern Computing. 5(1): 44-60.

[2] Pasham, S.D. (2017) AI-Driven Cloud Cost Optimization for Small and Medium Enterprises (SMEs). The Computertech. 1-24.

[3] Nersu, S., S. Kathram, and N. Mandaloju. (2020) Cybersecurity Challenges in Data Integration: A Case Study of ETL Pipelines. Revista de Inteligencia Artificial en Medicina. 11(1): 422-439.

[4] Technology Advancements (ESP-JETA). 1(1): 228-238.

[5] Pasham, S.D. (2018) Dynamic Resource Provisioning in Cloud Environments Using Predictive Analytics. The Computertech. 1-28.

[6] Mandaloju, N. kumar Karne, V., Srinivas, N., & Nadimpalli, SV (2021). Overcoming Challenges in Salesforce Lightning Testing with AI Solutions. ESP Journal of Engineering &

[7] Gudepu, B.K. and O. Gellago. (2018) Data Profiling, The First Step Toward Achieving High Data Quality. International Journal of Modern Computing. 1(1): 38-50.

[8] Reddy, V.M. and L.N. Nalla. (2024) Real-time Data Processing in E-commerce: Challenges and Solutions. International Journal of Advanced Engineering Technologies and Innovations. 1(3): 297-325.

[9] Nersu, S., S. Kathram, and N. Mandaloju. (2021) Automation of ETL Processes Using AI: A Comparative Study. Revista de Inteligencia Artificial en Medicina. 12(1): 536-559.

[10] Nadimpalli, S. Varma, and S. Noone. (2022) Strengthening Cybersecurity through Behavioral Analytics: Detecting Anomalies and Preventing Breaches. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence. 13(1): 243-258.

[11] Kumar Karne, V., N. Srinivas, N. Mandaloju, and S.V. Nadimpalli. (2023) Infrastructure as Code: Automating Multi-Cloud Resource Provisioning with Terraform. International Journal of Information Technology (IJIT). 9(1).