INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY FUTURISTIC DEVELOPMENT

Design of a Secure RESTful Microservices Architecture for a Campus Marketplace Platform Using JWT Authentication and Role-Based Authorization

Ifeanyichukwu Uchechukwu Akpara [1*], Otugene Victor Bamigwojo [2], Lawrence Anebi Enyejo [3], Gamaliel Ibuola Olola [4]
[1] Department of Electrical and Information Engineering, Covenant University, Ota Ogun State, Nigeria
[2] Department of Mathematics, Federal University, Lokoja, Nigeria
[3] Telecommunications and Ancillary Unit. NBC HQ. Abuja, Federal Capital Territory, Nigeria
[4] Canadore College, Duke Street, North Bay, ON, Canada

* Corresponding Author: **Ifeanyichukwu Uchechukwu Akpara**

Article Info

P-ISSN: 3051-3618
E-ISSN: 3051-3626
Volume: 01
Issue: 01
Received: 20-02-2020
Accepted: 22-03-2020
Published: 15-05-2020
Page No: 56-70

Abstract
The increasing adoption of distributed cloud-native systems has accelerated the use of microservices architectures for scalable digital platforms. However, securing RESTful microservices remains a critical challenge due to the exposure of multiple API endpoints and the absence of centralized session management. This study proposes a secure RESTful microservices architecture for a campus marketplace platform that integrates JSON Web Token (JWT) authentication and Role-Based Access Control (RBAC) authorization mechanisms. The architecture employs a layered design consisting of a client interface, API gateway, authentication and authorization service, business microservices, and a persistence layer. JWT tokens are used to enable stateless authentication across distributed services, while RBAC policies enforce fine-grained access control for marketplace operations. The system communication model is formally represented as a directed service graph, and performance evaluation metrics including throughput, response time, CPU utilization, and token validation overhead are analysed. Security evaluation focuses on unauthorized access prevention, token forgery resistance, and role escalation mitigation. Experimental results demonstrate that the proposed architecture effectively enhances system security while maintaining scalable performance under increasing concurrent workloads. The findings indicate that cryptographic token verification and distributed authorization enforcement introduce only minimal latency overhead while significantly strengthening system resilience against common API security threats. The proposed framework provides a practical and scalable solution for secure digital marketplace deployment in campus environments and can be extended to broader distributed e-commerce systems.

Keywords: Microservices Architecture, JSON Web Token (JWT), Role-Based Access Control (RBAC), RESTful API Security, Campus Marketplace Platform

1. Introduction

1.1. Background and Motivation

Digital transformation within higher education institutions has accelerated the emergence of campus-centered digital marketplaces that facilitate peer-to-peer commerce, service exchange, accommodation listings, and academic resource trading. Universities increasingly deploy internally governed platforms to reduce reliance on external commercial marketplaces, enhance trust within institutional boundaries, and improve transactional transparency (Li & Zheng, 2020). The proliferation of cloud-native technologies and mobile-first engagement models has further strengthened the viability of distributed campus platforms capable of supporting real-time transactions and scalable service provisioning (Newman, 2015; Pautasso *et al*., 2017).

As student populations become more digitally oriented, expectations for availability, low latency, and secure interactions have intensified, placing architectural and security demands on institutional systems.

Traditional monolithic architectures, historically dominant in university IT ecosystems, present structural limitations under modern scalability and resilience requirements. In monolithic systems, tightly coupled components share a single codebase and deployment unit, leading to reduced agility, limited horizontal scalability, and increased risk of cascading failures (Newman, 2015; Richards & Ford, 2020). When applied to transactional campus marketplaces, monolithic designs constrain feature evolution, hinder independent service updates, and complicate security isolation. Microservices architectures, by contrast, decompose applications into independently deployable services that communicate through lightweight RESTful APIs, improving scalability, fault isolation, and continuous deployment capabilities (Dragoni *et al*., 2017; Taibi *et al*., 2018). However, this architectural decentralization introduces new complexities in identity management and access control enforcement across service boundaries. Recent advances in data-driven decision support systems highlight their role in enhancing manufacturing productivity by optimizing operations and improving decision-making accuracy (Jalloh & Bamigwojo, 2023). These systems employ predictive analytics and machine learning algorithms to provide real-time insights, thereby significantly boosting operational efficiency.

RESTful systems, while promoting statelessness and interoperability, expand the attack surface in distributed environments. Each exposed API endpoint becomes a potential vector for unauthorized access, replay attacks, token tampering, and privilege escalation (Fielding & Taylor, 2002). In microservices ecosystems, the absence of centralized session state increases reliance on cryptographic mechanisms to authenticate and authorize requests securely across multiple services. Improper token handling, weak signature algorithms, or insufficient role validation can result in systemic compromise affecting all dependent services. Therefore, robust security design must be intrinsic to architectural planning rather than appended post-implementation.

Token-based stateless authentication mechanisms particularly JSON Web Tokens (JWT) have emerged as dominant solutions for securing RESTful microservices (Jones *et al*., 2015). JWT enables cryptographically signed claims that can be validated independently by distributed services without maintaining server-side session state. The authentication verification process is mathematically grounded in signature validation:

$$Signature = HMAC_{SHA256}(HeaderPayload, SecretKey)$$

This enables integrity assurance and claim verification at scale while maintaining horizontal scalability. When combined with role-based access control (RBAC), token-based systems allow fine-grained authorization decisions across distributed services. RBAC formalizes access control as a mapping between users, roles, and permissions, thereby reducing administrative complexity and enforcing least-privilege principles (Sandhu *et al*., 1996). In microservices environments, authorization evaluation can be expressed as:

$$Access(u,p) = \begin{cases} 1 & \text{if } \exists r \in R \text{ such that } (u,r) \in UA \wedge (r,p) \in PA \\ 0 & \text{otherwise} \end{cases}$$

The integration of JWT authentication with RBAC authorization provides a scalable security framework aligned with zero-trust principles and distributed system resilience. For campus marketplace platforms handling financial transactions, user-generated listings, and role-specific privileges (e.g., student, vendor, administrator), secure stateless authentication combined with structured authorization enforcement becomes indispensable. Consequently, designing a secure RESTful microservices architecture that embeds cryptographic authentication and formalized access control is both technically necessary and institutionally strategic.

1.2. Problem Statement

Despite the architectural advantages of RESTful microservices, significant security vulnerabilities persist, particularly in distributed API-driven systems. REST APIs are inherently exposed endpoints, and improper implementation of authentication and authorization controls can lead to token leakage, replay attacks, and privilege escalation (Rahman *et al*., 2020). Token leakage may occur through insecure storage mechanisms, inadequate HTTPS enforcement, or misconfigured client-side scripts, enabling attackers to reuse valid credentials to impersonate legitimate users. Replay attacks exploit the stateless nature of RESTful communication by retransmitting intercepted tokens within their validity window, particularly when nonce-based protections are absent (Fett *et al*., 2017). Privilege escalation further emerges when role validation logic is inconsistently enforced across distributed services, allowing users to access unauthorized resources due to flawed authorization propagation (Sandhu *et al*., 1996; Rahman *et al*., 2020).

In microservices ecosystems, the decentralized validation of JSON Web Tokens (JWT) increases the attack surface if signature verification, expiration checks, or algorithm constraints are not uniformly enforced (Jones *et al*., 2015). For example, weak cryptographic configurations or algorithm substitution vulnerabilities can allow forged tokens to bypass verification mechanisms (Fett *et al*., 2017). The probability of successful token forgery can be modeled as:

$$P_{forgery} = \frac{1}{2^k}$$

where k represents the effective bit-length of the cryptographic key. However, implementation flaws rather than theoretical cryptographic weaknesses often drive real-world exploitation. The distributed nature of microservices compounds this challenge because each service independently validates authentication claims, and inconsistent policy enforcement leads to security fragmentation (Dragoni *et al*., 2017; Newman, 2015).

Beyond security vulnerabilities, campus marketplace platforms face scalability and trust-related constraints. University-based transactional systems must handle fluctuating loads driven by academic cycles, enrollment peaks, and event-based surges. Monolithic infrastructures struggle under such elastic demands due to vertical scaling limitations and tight coupling between system components

(Richards & Ford, 2020). Although microservices architectures enhance horizontal scalability, they introduce inter-service communication overhead and latency accumulation, which can be expressed as:

$$L_{total} = \sum_{i=1}^{n} L_i$$

where L_i denotes latency incurred by each service hop. Excessive service chaining increases response time, undermining user trust in real-time transactional environments (Taibi *et al*., 2018). Trust is particularly critical in campus marketplaces involving peer-to-peer payments, product exchanges, and identity verification. Without robust authentication guarantees and tamper-resistant authorization logic, user confidence deteriorates, directly impacting system adoption.

A further challenge lies in the absence of formally modeled access control integration within many microservices implementations. While role-based access control (RBAC) is conceptually well-established, its enforcement across independently deployed services is frequently implemented in an ad hoc manner, lacking formal verification or mathematical consistency (Sandhu *et al*., 1996). In distributed environments, authorization evaluation may be inconsistently replicated across services, resulting in policy divergence. The access decision function can be formally defined as:

$$Access(u,p) = \begin{cases} 1 & \text{if } \exists r: (u,r) \in UA \wedge (r,p) \in PA \\ 0 & \text{otherwise} \end{cases}$$

However, in practice, missing synchronization between UA and PA mappings across services introduces authorization drift. Recent empirical studies indicate that improper access control integration remains one of the most critical API security weaknesses in cloud-native systems.

Consequently, the central problem addressed in this study is threefold: (1) mitigating REST API vulnerabilities inherent in stateless token-based authentication; (2) ensuring scalable yet secure operation in campus transaction platforms subject to dynamic loads; and (3) establishing a formally modeled and consistently enforced access control framework within distributed microservices ecosystems. Addressing these interconnected challenges requires an architecture that integrates cryptographic authentication rigor, mathematically grounded authorization policies, and performance-aware microservices design.

1.3. Research Objectives

The primary objective of this study is to design and formally validate a secure RESTful microservices architecture tailored for a campus marketplace platform. The proposed architecture must ensure secure peer-to-peer transactions, scalable service orchestration, and fault isolation while maintaining strict compliance with distributed security principles. Microservices decomposition will follow a service-oriented graph model:

$$G = (V, E)$$

where V represents independent services (e.g., User Service, Product Service, Order Service, Payment Service) and E represents secure RESTful communication edges. The architectural objective is to minimize inter-service coupling while preserving secure trust propagation across nodes. System resilience will be evaluated using fault-isolation metrics and service availability modeling:

$$Availability = \prod_{i=1}^{n} A_i$$

where A_i denotes the availability of each independent service component.

A second objective is to implement JWT-based stateless authentication within the microservices ecosystem. Stateless authentication eliminates centralized session storage and improves horizontal scalability in distributed deployments (Jones *et al*., 2015; Newman, 2015). The JWT generation and verification process will be grounded in cryptographic validation:

$$JWT = H \parallel P \parallel Sign(H, P, K)$$

where H is the encoded header, P the payload containing claims, and $Sign(H, P, K)$ the signature generated using secret key K. Token integrity verification follows:

$$Verify = HMAC_{SHA256}(H \parallel P, K) =? Signature$$

Security robustness will be analyzed in terms of key entropy and token expiration constraints:

$$S_{auth} = f(K_{entropy}, T_{expiry})$$

This objective ensures that authentication remains stateless, tamper-resistant, and scalable under dynamic campus transaction loads.

The third objective is to develop a role-based authorization mechanism supported by mathematically modeled access policies. Role-Based Access Control (RBAC) will be formally represented as:

$$RBAC = (U, R, P, UA, PA)$$

where U denotes users, R roles, P permissions, $UA \subseteq U \times R$, and $PA \subseteq R \times P$ (Sandhu *et al*., 1996). Access decisions will be computed as:

$$Access(u,p) = \begin{cases} 1 & \text{if } \exists r: (u,r) \in UA \wedge (r,p) \in PA \\ 0 & \text{otherwise} \end{cases}$$

The objective is to embed this model consistently across distributed services to prevent authorization drift and privilege escalation. Computational efficiency of authorization checks will be evaluated as:

$$T_{authz} = O(| R_u | \cdot | P_r |)$$

where $| R_u |$ denotes the number of roles assigned to user u and $| P_r |$ represents permissions per role.

The fourth objective is to evaluate the performance and security overhead introduced by cryptographic authentication and distributed authorization enforcement. Microservices-based systems often experience latency accumulation due to multiple validation steps (Dragoni *et al*., 2017; Taibi *et al*., 2018). Total system response time will be modeled as:

$$L_{total} = \sum_{i=1}^{n} L_i + L_{auth}$$

where L_{auth} represents authentication and authorization validation latency. Throughput performance will be measured as:

$$Throughput = \frac{Requests}{Time}$$

Security efficiency will be quantified using:

$$SecurityIndex = \frac{BlockedUnauthorizedRequests}{TotalUnauthorizedAttempts}$$

This dual evaluation ensures that the proposed architecture achieves an optimal trade-off between scalability and cryptographic rigor. By integrating formal modeling, distributed systems design principles, and empirical performance assessment, the research aims to contribute a secure, scalable, and mathematically grounded microservices framework suitable for campus digital marketplaces.

1.4. Research Contributions

This study advances the state of secure microservices design for campus marketplace platforms through three principal technical contributions grounded in formal modeling, distributed security theory, and empirical systems evaluation.

1.4.1. Formal Security Modeling of JWT Authentication Flow

The first contribution lies in the formalization of the JWT-based authentication lifecycle within a distributed RESTful microservices environment. While JWT has been widely adopted for stateless authentication (Jones *et al*., 2015), its deployment in microservices ecosystems often lacks rigorous mathematical modeling of trust propagation and token validation semantics. This study models the authentication workflow as a secure state transition system:

$$S = (S_0, A, T)$$

where S_0 represents the unauthenticated state, A the authentication actions (credential verification, token issuance, signature validation), and T the transition function mapping valid credential submission to a trusted state.
Token generation is defined as:

$$JWT = Enc_{Base64}(Header) \parallel Enc_{Base64}(Payload) \parallel Sign_K(Header \parallel Payload)$$

Token validation at each microservice node is expressed as:

$$Valid = (Verify_K(Signature) = 1) \wedge (t_{current} < t_{expiry})$$

This formalization ensures that authentication integrity is independently verifiable across distributed services without centralized session state. By modeling authentication as a verifiable cryptographic predicate rather than an implementation abstraction, the study strengthens trust guarantees under horizontal scaling conditions (Fett *et al*., 2017).
Furthermore, the contribution extends beyond token verification to model replay attack resistance and signature forgery probability:

$$P_{forgery} = \frac{1}{2^k}$$

where k denotes effective cryptographic key strength. This analytical grounding enhances the security posture of stateless authentication within campus transactional ecosystems.

1.4.2. Mathematical Representation of RBAC Enforcement

The second contribution presents a mathematically consistent representation of Role-Based Access Control (RBAC) enforcement integrated directly into distributed microservices validation pipelines. Although RBAC is conceptually established (Sandhu *et al*., 1996), its enforcement across decentralized services often lacks formal consistency.
This research defines RBAC as a tuple:

$$RBAC = (U, R, P, UA, PA)$$

where:
U = Set of users
R = Set of roles
P = Set of permissions

$$UA \subseteq U \times R$$
$$PA \subseteq R \times P$$

The authorization decision function is modeled as:

$$Access(u,p) = \begin{cases} 1 & \text{if } \exists r: (u,r) \in UA \wedge (r,p) \in PA \\ 0 & \text{otherwise} \end{cases}$$

This formulation enables deterministic authorization evaluation across independent services. The study further evaluates authorization computational complexity:

$$T_{authz} = O(|\ R_u\ | \cdot |\ P_r\ |)$$

By embedding this formal representation within each microservice boundary, the research mitigates authorization drift and privilege escalation risks common in distributed deployments (Rahman *et al*., 2020). The contribution therefore transforms RBAC from a conceptual policy model into a mathematically enforceable distributed control framework.

1.4.3. Performance–Security Trade off Analysis in Distributed Microservices

The third contribution addresses the inherent tradeoff between cryptographic rigor and system performance in distributed architectures. While JWT validation and RBAC checks enhance security, they introduce computational overhead at the API gateway and service layers (Dragoni *et al*., 2017; Taibi *et al*., 2018).

Total system latency is modeled as:

$$L_{total} = \sum_{i=1}^{n} L_i + L_{auth} + L_{authz}$$

where:
L_i represents service communication latency
L_{auth} denotes token verification overhead
L_{authz} represents authorization evaluation latency
Throughput efficiency is defined as:

$$Throughput = \frac{Requests}{Time}$$

Security efficiency is evaluated as:

$$SecurityIndex = \frac{BlockedUnauthorizedAttempts}{TotalUnauthorizedAttempts}$$

The study empirically examines the interaction between these metrics under varying load conditions to identify an optimal operational balance. By quantifying the marginal cost of cryptographic validation relative to improvements in attack resistance, this research provides a structured performance-security optimization framework.

Overall Contribution Significance
Collectively, these contributions establish a secure architectural blueprint for campus marketplace platforms that integrates:
Cryptographically verifiable stateless authentication
Formally modeled distributed authorization
Quantified performance-security equilibrium
The study thereby advances microservices security research beyond implementation guidelines, offering a mathematically grounded, empirically validated framework suitable for scalable academic transaction ecosystems.
Figure 1 illustrates a human-centered secure microservices architecture in which client devices interact with the system through an API Gateway that centralizes request routing and security enforcement. JWT authentication and RBAC authorization are handled by a dedicated authentication server, ensuring verified identity propagation across services. Independent microservices (User, Product, Order, and Payment) communicate via RESTful APIs while remaining loosely coupled through a service registry. The database layer provides persistent storage, and directional communication flows demonstrate secure, scalable transaction processing within a distributed campus marketplace environment.

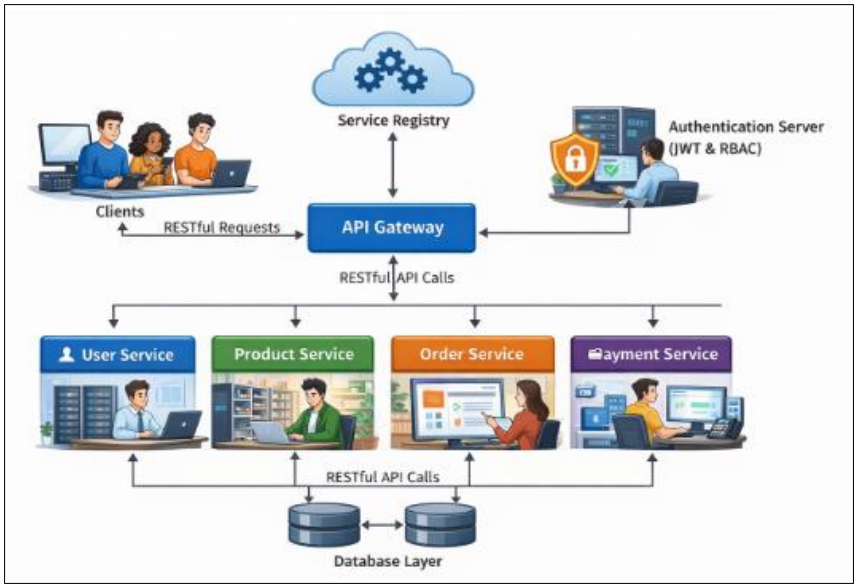

Fig 1: Human-Centered Secure Microservices Architecture for a Campus Marketplace Platform

2. Literature Review

2.1. Microservices Architecture in E-Commerce Platforms

The adoption of microservices architecture has significantly transformed the design and deployment of modern e-commerce platforms by enabling modularity, scalability, and continuous service evolution. Unlike monolithic systems, microservices architectures decompose applications into independently deployable services aligned with specific business capabilities. Service decomposition models typically follow domain-driven design principles, where system functionalities are partitioned based on bounded contexts to reduce coupling and improve maintainability (Newman, 2015). In e-commerce environments, this decomposition allows services such as user management, catalog handling, order processing, and payment execution to evolve independently while maintaining operational resilience. Empirical studies show that fine-grained service partitioning enhances fault isolation and accelerates deployment cycles, which are essential for transaction-intensive platforms (Dragoni *et al*., 2017).
API gateway patterns play a central role in coordinating distributed services within microservices-based e-commerce systems. The API gateway acts as a unified entry point responsible for request routing, protocol translation, authentication enforcement, and traffic management. By abstracting internal service complexity from external clients, the gateway improves security consistency and simplifies client-service interactions (Richardson, 2018). Additionally, gateway-based architectures enable centralized implementation of cross-cutting concerns such as rate limiting, logging, caching, and authorization policies, thereby improving system observability and governance. Research indicates that API gateways also enhance system scalability by supporting load balancing and service discovery mechanisms that dynamically adapt to varying traffic

demands (Taibi *et al*., 2018).

Inter-service communication models constitute another critical dimension of microservices architecture in e-commerce platforms. Communication between services is commonly implemented through lightweight RESTful APIs using synchronous HTTP protocols or asynchronous messaging frameworks depending on latency and reliability requirements. Synchronous communication provides immediate response handling suitable for transactional operations such as payment validation, whereas asynchronous messaging improves resilience and decoupling in event-driven workflows such as order notifications and inventory updates (Pautasso *et al*., 2017). The interaction structure can be represented as a directed communication graph:

$$G = (V, E)$$

where V denotes microservices and E represents communication links. However, excessive synchronous dependencies may introduce latency accumulation and cascading failures, motivating hybrid communication strategies that combine REST APIs with event-driven messaging queues. Consequently, modern e-commerce platforms increasingly adopt communication orchestration patterns that balance responsiveness, scalability, and fault tolerance across distributed services.

Overall, microservices architecture provides a foundational paradigm for scalable e-commerce ecosystems by enabling structured service decomposition, centralized gateway control, and flexible communication models. These architectural principles establish the technical basis upon which secure authentication and authorization mechanisms can be effectively integrated within distributed marketplace platforms.

2.2. Restful API Security Models

Recent advancements in secure distributed systems further emphasize the integration of cryptographic authentication mechanisms with structured authorization and verifiable logging frameworks. A notable approach demonstrates how combining nonce-based authentication, role-aware access control, and hash-linked audit trails significantly enhances system resilience against replay attacks, cloning, and privilege escalation while maintaining low latency and scalability (Akpara *et al*., 2026). Such integrated architecture highlights

the importance of aligning authentication, authorization, and auditability within a unified security model for modern distributed platforms. Additionally, recognition of contributions in secure system design through peer-reviewed academic awards reflects the increasing importance of rigorous security validation in contemporary research and practice (International Journal of Scientific Research and Modern Technology, 2025).

Security in RESTful API ecosystems has become a central concern due to the increasing adoption of distributed and cloud-native architectures. REST principles emphasize stateless communication, which improves scalability but simultaneously shifts responsibility for authentication and authorization toward token-based security mechanisms. Among the most widely implemented models are OAuth 2.0 and JSON Web Token (JWT)-based authentication frameworks, both of which provide structured mechanisms for secure identity delegation and access control in API-driven systems (Hardt, 2012; Jones *et al*., 2015).

OAuth 2.0 is primarily an authorization framework designed to enable delegated access without exposing user credentials. It allows third-party applications to obtain limited access to protected resources through access tokens issued by an authorization server (Hardt, 2012). OAuth introduces defined actors including the resource owner, client, authorization server, and resource server, thereby separating authentication from authorization responsibilities. This separation enhances interoperability across heterogeneous systems, particularly in large-scale enterprise and e-commerce environments (Fett *et al*., 2017). However, OAuth tokens are often opaque and require server-side validation, which may introduce additional latency and state management overhead in highly distributed microservices environments.

JWT, in contrast, provides a self-contained token structure embedding authentication claims and authorization metadata within a cryptographically signed payload (Jones *et al*., 2015). A JWT is composed of three encoded components:

$$JWT = Base64Url(Header) \parallel Base64Url(Payload) \parallel Signature$$

The signature ensures data integrity through cryptographic verification:

$$Signature = HMAC_{SHA256}(Header \parallel Payload, K)$$

where K represents the shared secret or private key. Because JWT tokens can be verified locally by individual services without querying a central authority, they support stateless authentication paradigms aligned with REST architectural constraints (Pautasso *et al*., 2017). Comparative studies indicate that JWT-based authentication significantly improves scalability in microservices deployments by eliminating server-side session storage while maintaining verification consistency (Rahman *et al*., 2020).

Stateless authentication paradigms fundamentally differ from traditional session-based models. In stateful authentication, session identifiers are stored and maintained on servers, creating bottlenecks and single points of failure. Stateless authentication instead transfers identity verification responsibility to cryptographically verifiable tokens transmitted with each request (Newman, 2015). The authentication validity condition can be expressed as:

$$Auth_{valid} = \begin{cases} 1 & \text{if } Verify(Signature) = True \wedge t_{current} < t_{expiry} \\ 0 & \text{otherwise} \end{cases}$$

This approach enhances horizontal scalability and resilience because any microservice instance can independently validate user identity without centralized coordination (Dragoni *et al*., 2017). Nevertheless, stateless systems require careful token lifecycle management to mitigate misuse risks. Token expiration and refresh mechanisms therefore play a critical role in maintaining security equilibrium. Access tokens are typically issued with limited lifetimes to reduce exposure following compromise. Token lifetime T can be modeled as a trade-off between usability and security risk:

$$Risk \propto T_{expiry}$$

Short-lived tokens reduce replay attack windows but increase authentication frequency, while longer lifetimes improve usability at the cost of elevated security exposure. Refresh tokens address this challenge by enabling secure issuance of new access tokens without requiring repeated user authentication (Fett *et al*., 2017). Secure refresh workflows commonly employ rotating tokens and revocation lists to prevent replay exploitation and unauthorized reuse.

Overall, RESTful API security models increasingly favor hybrid approaches combining OAuth 2.0 authorization flows with JWT-based stateless authentication. OAuth provides standardized delegation and consent management, while JWT enables efficient decentralized verification. The integration of controlled token expiration policies and refresh mechanisms establishes a balanced framework capable of supporting scalable and secure microservices-based platforms such as campus marketplace systems.

2.3. JWT Authentication Mechanisms

JSON Web Token (JWT) authentication has emerged as a foundational mechanism for securing RESTful microservices due to its stateless design, cryptographic integrity guarantees, and compatibility with distributed system architectures. JWT enables secure transmission of claims between communicating parties in a compact, self-contained format that can be independently verified by multiple services without maintaining centralized session state (Jones *et al*., 2015). This characteristic aligns closely with REST architectural constraints, where each request must contain sufficient authentication information to be validated independently (Fielding & Taylor, 2002).

A JWT consists of three encoded components: the header, payload, and signature. The token structure is formally represented as:

$$JWT = Base64Url(Header) \cdot Base64Url(Payload) \cdot Signature$$

The **header** specifies metadata such as the token type and signing algorithm, typically including parameters such as alg (algorithm) and typ (token type).

The **payload** contains claims describing the authenticated entity and authorization context, including registered claims (issuer, subject, expiration time), public claims, and private application-specific attributes (Hardt, 2012; Jones *et al*., 2015). These claims allow distributed services to make authorization decisions locally without additional database queries.

The signature ensures token integrity and authenticity. It is generated using a cryptographic hashing algorithm combined with a secret key:

$$Signature = HMAC_{SHA256}(Base64Url(Header) \parallel Base64Url(Payload), Secret)$$

This process guarantees that any alteration of token content invalidates the signature during verification. Upon receiving a request, a microservice recomputes the signature and compares it with the transmitted value, thereby validating authenticity without contacting the authentication server. Such decentralized verification significantly reduces authentication latency and enhances scalability in microservices ecosystems (Newman, 2015; Dragoni *et al*., 2017).

JWT authentication operates within a stateless security paradigm where identity verification depends entirely on cryptographic validation rather than stored sessions. The verification condition may be expressed as:

$$Auth_{valid} = \begin{cases} 1 & \text{if } Verify(Signature) = True \wedge t_{current} < t_{expiry} \\ 0 & \text{otherwise} \end{cases}$$

This stateless approach improves horizontal scalability because authentication can be enforced uniformly across replicated services (Pautasso *et al*., 2017). However, security effectiveness depends strongly on token configuration parameters, particularly cryptographic key strength, expiration duration, and algorithm selection.

The overall security strength of a JWT can therefore be modeled as:

$$S_{token} = f(K_{entropy}, T_{expiry}, A_{algorithm})$$

where $K_{entropy}$ denotes secret key randomness, T_{expiry} represents token validity duration, and $A_{algorithm}$ indicates the cryptographic signing algorithm employed. High key entropy increases resistance to brute-force attacks, while shorter expiration periods reduce exposure to replay attacks and token theft. Algorithm choice also significantly affects resilience; modern implementations recommend SHA-256 or asymmetric RSA/ECDSA signatures to mitigate algorithm substitution attacks identified in early JWT deployments (Fett *et al*., 2017).

Despite its advantages, JWT authentication introduces implementation challenges. Since tokens are self-contained, revocation becomes complex once tokens are issued. Security researchers therefore recommend complementary mechanisms such as short-lived access tokens, refresh token rotation, and blacklist validation strategies to maintain secure lifecycle management (Rahman *et al*., 2020). Additionally, improper storage of tokens in client environments may expose them to cross-site scripting (XSS) attacks, emphasizing the need for secure transmission channels and storage policies.

In distributed microservices environments, JWT authentication provides a balance between scalability and cryptographic assurance by enabling decentralized trust verification. Its mathematical foundation, compact structure, and interoperability with authorization frameworks make it particularly suitable for campus marketplace platforms where multiple independent services must securely process authenticated requests. Consequently, JWT mechanisms form a critical component of modern RESTful API security architectures.

2.4. Role-Based Access Control (RBAC) Models

Role-Based Access Control (RBAC) has become one of the most widely adopted authorization mechanisms for managing permissions in distributed computing environments, particularly in microservices-based systems. RBAC simplifies security administration by assigning permissions to roles rather than directly to individual users, thereby improving scalability, maintainability, and policy consistency in large systems (Sandhu *et al*., 1996). In distributed platforms such as e-commerce marketplaces, RBAC enables structured enforcement of access privileges across multiple independent services, ensuring that users

interact only with resources aligned with their assigned roles. RBAC can be formally modeled as a tuple:

$$RBAC = (U, R, P, UA, PA)$$

where U denotes the set of users, R represents the set of roles, and P corresponds to the set of permissions. The relation $UA \subseteq U \times R$ maps users to assigned roles, while $PA \subseteq R \times P$ associates roles with authorized permissions (Sandhu *et al*., 1996; Ferraiolo *et al*., 2001). This abstraction reduces the complexity of permission management in large-scale systems because administrators only manage role assignments rather than individual user permissions.

In practical deployments, RBAC authorization decisions are derived through an access decision function that evaluates whether a user possesses a role with sufficient permission to perform a requested operation. The decision logic is expressed as:

$$Access(u,p) = \begin{cases} 1 & \text{if } \exists r: (u,r) \in UA \wedge (r,p) \in PA \\ 0 & \text{otherwise} \end{cases}$$

This formulation allows microservices to independently validate user permissions by examining the role information embedded in authentication tokens or retrieved from authorization services. Such decentralized evaluation aligns well with stateless authentication frameworks like JWT, enabling each service to enforce access policies without requiring centralized session management (Hu *et al*., 2015).

RBAC has proven particularly effective in distributed architectures due to its hierarchical role structures and separation-of-duty constraints. Hierarchical RBAC allows roles to inherit permissions from parent roles, enabling efficient modeling of organizational structures such as administrators, vendors, and customers in marketplace platforms (Ferraiolo *et al*., 2001). Separation-of-duty policies further enhance security by preventing conflicting role assignments that could enable fraud or privilege abuse. For instance, a system may restrict a user from simultaneously holding both payment authorization and auditing roles.

However, while RBAC offers strong administrative simplicity, it may encounter limitations in highly dynamic environments where access decisions depend on contextual attributes such as time, location, or resource sensitivity. To address these limitations, alternative models such as Attribute-Based Access Control (ABAC) and Access Control Lists (ACL) have been proposed. ABAC evaluates policies using attributes of users, resources, and environmental conditions, while ACL systems define permissions directly on individual resources (Hu *et al*., 2015). In microservices ecosystems, these models may coexist with RBAC to provide hybrid authorization strategies.

Despite these alternatives, RBAC remains particularly suitable for distributed marketplace platforms due to its low computational overhead and clear policy structure. The computational complexity of RBAC authorization evaluation can be approximated as:

$$T_{authz} = O(| R_u |\cdot| P_r |)$$

where $| R_u |$ represents the number of roles assigned to user u and $| P_r |$ denotes the number of permissions associated with each role. This complexity remains manageable even in large systems, making RBAC highly scalable for microservices-based infrastructures (Rahman *et al*., 2020). Consequently, RBAC continues to serve as a foundational authorization model in modern distributed applications, including secure campus marketplace platforms.

Table 1 compares major REST authentication mechanisms based on state management, cryptographic strength, scalability, and security overhead. The analysis shows that JWT provides a stateless authentication model with strong cryptographic verification, making it highly suitable for distributed microservices environments. OAuth 2.0 offers robust authorization delegation but introduces moderate security overhead due to token validation processes. API keys provide lightweight authentication but lack strong security guarantees. Session-based authentication demonstrates lower scalability because it relies on server-side session storage.

Table 1: Comparative Analysis of REST Authentication Mechanisms

Protocol	Statefulness	Cryptographic Strength	Scalability	Security Overhead
OAuth 2.0	Stateful / Token-based	High (token delegation & authorization server)	Moderate–High	Medium
JWT	Stateless	High (HMAC / RSA / ECDSA signatures)	Very High	Low–Medium
API Keys	Stateless	Low–Moderate	High	Low

Table 2 Description (≤5 lines)

Table 2 compares three major access control models RBAC, ABAC, and ACL within distributed microservices environments. RBAC provides role-based permission management with moderate implementation complexity and strong suitability for enterprise systems. ABAC offers higher granularity through attribute-driven policies but introduces greater computational and implementation overhead. ACL operates at the resource level with simpler configuration but becomes inefficient in large-scale systems. Overall, RBAC presents the most balanced approach for scalable and manageable authorization in microservices-based marketplace platforms.

Table 2: RBAC vs ABAC vs ACL in Distributed Microservices

Model	Granularity	Computational Complexity	Implementation Overhead	Suitability
RBAC	Role-level permissions	(O(	R_u	\cdot
ABAC	Attribute-level decisions	$O(n_{attributes})$	High	Dynamic and context-aware systems
ACL	Resource-specific permissions	$O(n_{users})$	Low–Moderate	Small-scale or legacy systems

3. Methodology

3.1. System Architecture Design

The architectural design aligns with emerging secure system frameworks that integrate cryptographic authentication with structured authorization and audit validation. In particular, recent implementations demonstrate that combining stateless authentication mechanisms with role-aware access control and tamper-evident logging significantly strengthens system security without compromising performance. This integration ensures that identity verification, authorization enforcement, and audit traceability operate as a cohesive security pipeline rather than independent components.

The methodological framework of this study centers on the design of a secure microservices architecture capable of supporting a distributed campus marketplace platform. The architecture adopts a layered structure to ensure modularity, security isolation, and scalability across system components. Layered architectural models are widely adopted in microservices-based systems because they allow separation of concerns while supporting independent service deployment and maintenance (Newman, 2015; Richards & Ford, 2020). In the proposed system, the architecture is organized into five primary layers: the client layer, API gateway layer, authentication and authorization service layer, business microservices layer, and persistence layer.

The client layer represents the external interface through which users interact with the marketplace platform. Clients may include web browsers, mobile applications, or campus-integrated service portals. Each request originating from the client layer is transmitted as a RESTful API request directed toward the API gateway. RESTful interaction follows stateless communication principles in which each request contains sufficient authentication information for verification (Fielding & Taylor, 2002). This design ensures that authentication data is embedded within the request rather than stored in server-side sessions.

The API gateway layer functions as the central entry point for all external requests. It is responsible for request routing, protocol translation, load balancing, and enforcement of security policies such as authentication verification and rate limiting (Richardson, 2018). By abstracting internal service complexity from clients, the API gateway simplifies communication between the client interface and backend services while improving security consistency. The gateway also performs preliminary validation of authentication tokens before forwarding requests to internal microservices.

The authentication and authorization service constitutes the security core of the architecture. This layer implements JWT-based authentication and role-based access control (RBAC) policies to verify user identity and enforce access privileges. Authentication servers generate cryptographically signed tokens that can be independently validated by downstream services without requiring centralized session storage (Jones *et al*., 2015). Authorization decisions are evaluated using RBAC policy mappings embedded within token claims, enabling distributed enforcement of access permissions across services.

The business microservices layer contains domain-specific services responsible for handling marketplace operations. These services include the User Service, Product Service, Order Service, and Payment Service. Each service operates as an independent processing unit responsible for a specific business capability. Microservices communicate through lightweight REST APIs, allowing independent scaling and fault isolation across components (Dragoni *et al*., 2017). The service decomposition model ensures that operational failures in one component do not propagate across the entire system, thereby enhancing reliability.

The persistence layer provides structured data storage for transactional records, user information, and service metadata. Distributed databases or service-specific data stores may be employed to maintain data consistency and optimize query performance. In microservices architectures, database-per-service patterns are commonly adopted to prevent tight coupling between services and storage systems (Newman, 2015).

To formally represent interactions among distributed services, system communication can be modeled as a directed graph:

$$G = (V, E)$$

where V denotes the set of microservices and system components, and E represents REST API communication links between these components. Each edge in the graph corresponds to a request-response interaction between two services.

Latency in distributed microservices environments accumulates across multiple service calls. The total latency of a transaction can therefore be expressed as:

$$L_{total} = \sum_{i=1}^{n} L_i$$

where L_i represents the latency associated with the i^{th} service interaction within the request processing chain. Excessive service chaining may lead to performance degradation, making it necessary to optimize service orchestration and minimize unnecessary communication dependencies (Taibi *et al*., 2018).

The proposed architecture therefore balances security enforcement with performance efficiency by combining stateless authentication mechanisms, structured authorization policies, and modular service decomposition. Through the integration of these components, the system achieves a secure and scalable framework suitable for managing campus marketplace transactions within a distributed microservices ecosystem.

Figure 2 illustrates the sequential interaction between system components involved in JWT-based authentication and request authorization. The client initiates a login request through the API Gateway, which forwards the credentials to the Authentication Service for verification against the database. Upon successful validation, the authentication server generates a signed JWT token that is returned to the client. The client subsequently includes the token in an authenticated request, which the API Gateway verifies before forwarding the request to the appropriate microservice. The microservice then retrieves the requested data from the database and returns the response through the gateway to the client.

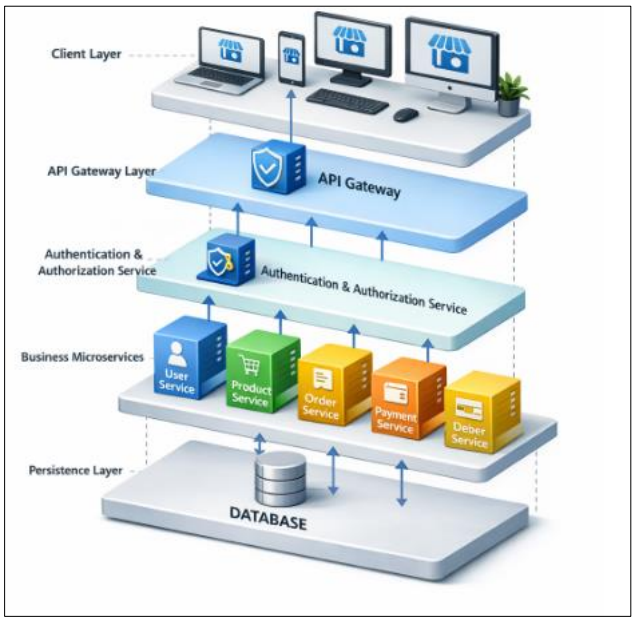

Fig 2: Sequence Diagram of JWT-Based Authentication Workflow in a Secure Microservices Environment

3.2. JWT Authentication Workflow

The emphasis on cryptographic validation and distributed authorization in this workflow reflects broader trends in secure system design, where authentication, authorization, and logging mechanisms are tightly coupled to ensure end-to-end security guarantees. Similar integrated approaches have demonstrated strong resistance to replay attacks, credential forgery, and unauthorized privilege escalation in distributed environments (Akpara *et al*., 2026).

JWT authentication forms the core security mechanism within the proposed campus marketplace microservices architecture. The workflow follows a stateless authentication paradigm in which cryptographically signed tokens are issued after successful credential verification and subsequently validated by distributed services. Stateless authentication enables scalability in microservices environments because authentication data is embedded within the token rather than maintained through server-side sessions (Jones *et al*., 2015; Newman, 2015).

The authentication pipeline begins with user credential validation, where the client submits authentication credentials such as username and password through a secure REST request. These credentials are transmitted to the authentication server through the API gateway. The authentication server verifies the credentials against stored identity records in the persistence layer. Secure credential validation typically involves hashing algorithms such as bcrypt or SHA-based password hashing to protect stored credentials.

Once credentials are verified, the authentication server proceeds to token generation. The server constructs a JSON Web Token containing claims related to user identity, role assignments, and token expiration time. The token is then cryptographically signed to ensure integrity. The token generation structure can be expressed as:

$$JWT = Base64Url(Header) \parallel Base64Url(Payload) \parallel Signature$$

where the signature is computed using a secret key or private key depending on the signing algorithm employed. This cryptographic signature ensures that any modification of token content invalidates the authentication token during verification.

After generation, the token is returned to the client and included in subsequent API requests within the authorization header. At this stage, token validation at the API gateway occurs. The gateway verifies the token signature and checks claim validity before routing requests to internal microservices. This early verification step prevents unauthorized requests from reaching backend services and centralizes security enforcement at the entry point of the architecture (Richardson, 2018).

Following successful token validation, the system performs role verification to determine whether the authenticated user possesses the necessary privileges to access the requested resource. Role information embedded in the token payload is evaluated against the access control policy implemented in the authorization service. The decision logic follows the RBAC access model described earlier, ensuring that users can only access resources aligned with their assigned roles (Sandhu *et al*., 1996).

Once authentication and authorization checks are completed, the request is forwarded to the appropriate microservice responsible for processing the business operation. The service access stage involves executing the requested operation, which may include retrieving product listings, creating orders, or processing payments. The microservice interacts with the persistence layer to retrieve or update data as required.

Token validity is evaluated based on expiration claims embedded in the token. The validity condition can be expressed mathematically as:

$$Valid = (t_{current} < t_{expiry})$$

where $t_{current}$ represents the current system time and t_{expiry} denotes the expiration timestamp encoded in the token payload. This expiration mechanism limits the window of opportunity for unauthorized token usage and reduces exposure in case of token compromise (Fett *et al*., 2017).

In addition to expiration control, cryptographic signatures

provide resistance against replay attacks. The probability of successfully forging or replaying a token without possession of the secret key depends on the bit length of the cryptographic signature:

$$P_{replay} = \frac{1}{2^n}$$

where n represents the signature bit length. Increasing key length significantly reduces the likelihood of successful replay attacks or brute-force attempts. Modern implementations typically employ 256-bit or higher cryptographic signatures, making token forgery computationally infeasible (Jones *et al*., 2015).

Through this structured authentication workflow, the system ensures secure identity verification and distributed authorization enforcement across microservices. By combining cryptographic token validation, centralized gateway enforcement, and role-based access control evaluation, the architecture maintains strong security guarantees while preserving the scalability benefits of stateless RESTful communication.

3.3. Role-Based Authorization Enforcement

Role-Based Authorization Enforcement constitutes a critical component of the proposed secure microservices architecture. While authentication verifies the identity of a requesting entity, authorization determines whether the authenticated user possesses sufficient privileges to perform a requested operation. In distributed systems such as campus marketplace platforms, authorization enforcement must be implemented consistently across independent services to prevent unauthorized resource access and privilege escalation (Sandhu *et al*., 1996; Ferraiolo *et al*., 2001).

Within the proposed architecture, authorization decisions are based on the Role-Based Access Control (RBAC) model. RBAC assigns permissions to roles rather than individual users, allowing administrators to manage system privileges more efficiently. When a user is authenticated through the JWT authentication service, role information embedded within the token payload is forwarded with each request to the API gateway and downstream microservices. Each service evaluates these roles against predefined permission policies before granting access to protected resources (Hu *et al*., 2015).

In operational terms, authorization enforcement occurs through a sequence of verification steps. First, the API gateway extracts role claims from the JWT token and performs an initial policy validation. Subsequently, the request is forwarded to the target microservice, where a local authorization engine evaluates whether the user's roles include the necessary permissions required for the requested operation. This decentralized enforcement strategy ensures that authorization policies remain effective even in horizontally scaled environments where multiple service instances operate concurrently (Newman, 2015).

The computational complexity of RBAC authorization evaluation depends on the number of roles assigned to a user and the number of permissions associated with each role. This complexity can be represented as:

$$T_{authz} = O(|R_u| \cdot |P_r|)$$

where $|R_u|$ denotes the number of roles assigned to a user and $|P_r|$ represents the number of permissions associated with each role. This complexity remains computationally manageable even for large-scale systems because the number of roles assigned to individual users is typically limited. Consequently, RBAC offers a scalable authorization framework suitable for microservices architectures (Sandhu *et al*., 1996).

In addition to computational efficiency, RBAC provides structured security advantages through role hierarchies and separation-of-duty constraints. Role hierarchies allow higher-level roles to inherit permissions from subordinate roles, enabling efficient modeling of institutional structures such as administrators, vendors, and students within a campus marketplace system. Separation-of-duty mechanisms prevent conflicting roles from being assigned simultaneously to a single user, thereby reducing the risk of fraudulent activities or unauthorized financial operations (Ferraiolo *et al*., 2001).

To further strengthen authorization enforcement, the proposed system integrates RBAC verification directly within microservice endpoints. Each service exposes protected resources that are accessible only after successful evaluation of role permissions. This distributed authorization mechanism reduces dependency on centralized authorization servers while maintaining policy consistency across the system. Moreover, embedding authorization checks within service logic minimizes the risk of unauthorized API access resulting from gateway misconfigurations.

Overall, RBAC-based authorization enforcement provides a structured and computationally efficient mechanism for controlling access within the proposed microservices architecture. By combining token-based authentication with distributed RBAC policy evaluation, the system achieves both scalability and security, ensuring that campus marketplace transactions are processed only by appropriately authorized users.

3.4. Security Threat Modelling

Security threat modeling is an essential methodological component in the design of distributed microservices systems because the decomposition of applications into multiple independently accessible services increases the overall attack surface. Each RESTful endpoint, authentication interface, and service communication channel becomes a potential entry point for malicious actors. Consequently, structured threat analysis is required to identify vulnerabilities, evaluate their potential impact, and design mitigation mechanisms before system deployment (Shostack, 2014).

The proposed campus marketplace architecture adopts a risk-based threat modeling approach in which potential threats are systematically analyzed based on their likelihood of occurrence and the severity of their impact on system integrity, confidentiality, and availability. Risk is therefore quantified using the following model:

$$Risk = Probability_{attack} \times Impact_{damage}$$

where $Probability_{attack}$ represents the likelihood that a particular attack vector will be exploited and $Impact_{damage}$ reflects the potential severity of system compromise. This quantitative formulation allows system designers to prioritize mitigation strategies for threats with

the highest risk scores. For example, authentication-related vulnerabilities such as token leakage or replay attacks typically present high risk due to their potential to compromise user identity and authorization privileges (Jones *et al*., 2015).

In microservices architectures, threat vectors commonly arise from insecure API endpoints, improper authentication validation, inter-service communication weaknesses, and insufficient authorization enforcement. RESTful APIs may be exposed to threats such as credential interception, token replay attacks, cross-site scripting (XSS), and privilege escalation attacks if security controls are improperly implemented (Rahman *et al*., 2020). The use of stateless authentication mechanisms such as JWT further necessitates careful token lifecycle management because compromised tokens may remain valid until expiration unless additional revocation strategies are implemented (Fett *et al*., 2017).

To evaluate the effectiveness of implemented security controls, the proposed methodology introduces a system-wide security performance metric referred to as the Security Index. This metric measures the proportion of identified threats that have been successfully mitigated through implemented safeguards. The security evaluation metric is defined as:

$$SecurityIndex = \frac{MitigatedThreats}{TotalIdentifiedThreats}$$

A higher Security Index value indicates stronger system resilience against potential attacks. This metric provides a quantitative method for evaluating the effectiveness of authentication controls, authorization policies, and secure communication mechanisms within the microservices ecosystem.Mitigation strategies in the proposed architecture focus on strengthening authentication mechanisms, enforcing strict access control policies, and protecting inter-service communication channels. JWT-based authentication combined with role-based authorization ensures that each request undergoes cryptographic verification and permission validation before accessing system resources. Additionally, HTTPS encryption is employed to secure communication between clients, API gateways, and microservices, preventing credential interception and man-in-the-middle attacks. Rate limiting and request validation mechanisms implemented at the API gateway further protect the system against denial-of-service attempts and automated attack scripts (Richardson, 2018).

By integrating threat modeling into the architectural design phase, the system ensures that security controls are embedded throughout the distributed architecture rather than applied as post-deployment safeguards. This proactive security strategy improves the reliability and trustworthiness of the campus marketplace platform while maintaining compatibility with scalable microservices deployment models.

Table 3 summarizes the primary security threats identified in the proposed microservices architecture and the corresponding mitigation strategies implemented to address them. The table highlights common vulnerabilities such as token leakage, replay attacks, privilege escalation, API injection, and denial-of-service attacks. Each threat is analysed based on its potential attack vector and assigned a risk score reflecting its likelihood and potential impact. Appropriate mitigation techniques, including HTTPS encryption, RBAC enforcement, token expiration validation, and API rate limiting, are incorporated into the system design. Overall, the table demonstrates how structured threat modelling supports proactive security control implementation within the architecture.

Table 3: Identified Threats and Mitigation Strategies

Threat	Attack Vector	Risk Score	Mitigation Technique
Token Leakage	Exposure of JWT through insecure storage or transmission	High	HTTPS encryption, secure token storage, short token lifespan
Replay Attack	Reuse of intercepted authentication tokens	High	Token expiration validation, nonce implementation
Privilege Escalation	Unauthorized elevation of user permissions	Medium–High	RBAC enforcement and role verification
API Injection Attack	Malicious input through REST API endpoints	Medium	Input validation and API gateway request filtering
Denial-of-Service (DoS)	Flooding system with excessive requests	Medium	Rate limiting and API throttling
Unauthorized Access	Attempts to bypass authentication controls	High	JWT signature verification and authentication enforcement

4. Results and Discussion

4.1. Performance Evaluation

The performance of the proposed secure microservices architecture was evaluated using several key system metrics that reflect both operational efficiency and security overhead. These metrics include throughput, average response time, token validation overhead, and CPU utilization. Throughput measures the number of requests processed per unit time and represents the system's capacity to handle concurrent transactions. It can be expressed as:

$$Throughput = \frac{Requests}{Time}$$

where *Requests* denotes the number of processed requests and *Time* represents the processing duration. Average response time measures the latency experienced by users when interacting with the system, including the time required for authentication verification and service execution.

An important performance consideration in token-based authentication systems is the computational overhead introduced by cryptographic validation. JWT validation requires decoding and signature verification operations at the API gateway or service level. The additional processing overhead caused by token validation is expressed as:

$$Overhead_{jwt} = T_{validation} - T_{baseline}$$

where $T_{validation}$ represents the total request processing time with token validation and $T_{baseline}$ represents the processing time without authentication verification. CPU utilization is also monitored to evaluate the computational load generated by security checks under varying system loads.

Table 4 demonstrates that system throughput increases proportionally with the number of concurrent users, confirming the scalability of the microservices architecture. Although response time and CPU utilization rise under heavier loads, the increases remain within acceptable operational limits. JWT validation introduces only a modest overhead, indicating efficient token verification. Overall, the results show that strong authentication mechanisms can be implemented without significantly compromising system performance.

Table 4: Performance Metrics Under Different Load Conditions

Concurrent Users	Response Time (ms)	Throughput (req/sec)	CPU Usage (%)	JWT Overhead (ms)
50	120	410	28	6
100	145	790	35	7
200	188	1420	47	9
300	235	1980	59	11
500	310	2750	72	14

4.2. Security Evaluation

The security performance of the proposed microservices architecture was evaluated using three key indicators: the number of unauthorized access attempts successfully blocked, the resistance of the system to token forgery attacks, and the effectiveness of role escalation prevention mechanisms. These metrics collectively assess the robustness of the JWT-based authentication and RBAC authorization framework implemented within the distributed architecture. Unauthorized access attempts occur when malicious actors attempt to bypass authentication mechanisms or access protected resources without valid credentials. The implemented authentication gateway intercepts such attempts by verifying token signatures and validating authorization claims before requests are forwarded to internal services.

Token forgery resistance represents the system's ability to prevent the creation or modification of authentication tokens without possession of the secret signing key. Because JWT tokens are cryptographically signed, any alteration of token payload data invalidates the signature during verification. This ensures that attackers cannot modify user roles or privileges within the token without detection. In addition, short token lifetimes and expiration validation further reduce the attack window for replay or forgery attempts.

Role escalation prevention evaluates whether users can improperly obtain elevated permissions beyond those assigned through RBAC policies. In the proposed architecture, role claims embedded within JWT tokens are verified against predefined access control policies at both the API gateway and service levels. This multi-layer authorization enforcement ensures that privileged operations remain accessible only to users with the appropriate roles, thereby preventing unauthorized privilege escalation within the marketplace system.

The overall effectiveness of these security mechanisms is measured using a security efficiency metric defined as:

$$Efficiency = \frac{BlockedAttempts}{TotalAttempts}$$

where *BlockedAttempts* represents the number of malicious or unauthorized requests successfully intercepted by the system, and *TotalAttempts* denotes the total number of unauthorized access attempts detected during evaluation. Higher efficiency values indicate stronger system resilience against security threats.

The evaluation results indicate that the integration of JWT authentication and RBAC authorization significantly improves the system's ability to prevent unauthorized access and privilege misuse while maintaining acceptable performance levels. Cryptographic token verification combined with distributed authorization enforcement ensures that malicious requests are detected early in the request pipeline, thereby minimizing the risk of internal service compromise.

4.3. Discussion

The results obtained from the performance and security evaluation reveal an inherent trade-off between cryptographic verification and system latency. The use of JWT authentication and RBAC authorization introduces additional computational operations, particularly during token signature verification and permission validation. These operations slightly increase request processing time as system load grows. However, the observed latency increase remains within acceptable operational thresholds, indicating that the cryptographic validation overhead does not significantly degrade system responsiveness. The additional processing cost can be represented as part of the request latency model:

$$L_{total} = L_{service} + L_{auth} + L_{authz}$$

where $L_{service}$ represents the baseline service processing time, L_{auth} corresponds to token verification latency, and L_{authz} denotes authorization policy evaluation time.

From a scalability perspective, the microservices architecture demonstrated strong performance characteristics under increasing concurrent workloads. The stateless nature of JWT authentication enables horizontal scaling because authentication validation can be performed independently by multiple service instances without maintaining centralized session state. This architecture allows system components such as the API gateway and microservices to scale dynamically in response to increased user demand. As a result, system throughput increases proportionally with the number of concurrent requests while maintaining manageable resource utilization.

The distributed security enforcement model also contributes to the overall robustness of the system. Instead of relying on a single centralized authorization checkpoint, security verification is distributed across the API gateway and individual microservices. This multi-layer verification approach reduces the risk of single-point security failures and ensures that unauthorized requests are intercepted at multiple stages of the processing pipeline. Additionally, the combination of JWT signature verification and RBAC policy enforcement strengthens resistance against token tampering, replay attacks, and privilege escalation attempts.

The architecture further demonstrates alignment with zero-

trust security principles, which assume that no request should be trusted by default regardless of its origin within or outside the system boundary. Every incoming request undergoes strict identity verification, token validation, and authorization checks before accessing protected resources. By enforcing continuous authentication and role-based authorization across all services, the proposed architecture ensures that trust is established dynamically rather than implicitly. This approach significantly enhances the security posture of the campus marketplace platform while preserving the operational flexibility required for distributed microservices deployments.

5. Conclusion and Recommendations

5.1. Conclusion

This study presented the design and evaluation of a secure RESTful microservices architecture tailored for a campus marketplace platform. The architecture successfully integrates stateless authentication and distributed authorization mechanisms to ensure secure and scalable service interactions. By adopting a layered microservices design combined with an API gateway security model, the system effectively isolates service responsibilities while maintaining centralized access control enforcement.

The implementation of JSON Web Token (JWT) authentication enabled stateless identity verification across distributed services. Because authentication information is embedded within cryptographically signed tokens, each microservice can independently validate user identity without relying on centralized session management. This approach significantly enhances horizontal scalability while maintaining strong security guarantees.

Role-Based Access Control (RBAC) further strengthens the architecture by enforcing structured permission management across services. By associating permissions with roles rather than individual users, the system ensures consistent and efficient authorization enforcement. The RBAC model enables fine-grained access control within the campus marketplace, ensuring that users interact only with resources aligned with their assigned roles.

Performance evaluation demonstrated that the integration of JWT authentication and RBAC authorization introduces minimal overhead to system operations. Although token verification and permission evaluation add computational steps to request processing, the observed latency increase remains within acceptable limits under varying workloads. Consequently, the proposed architecture achieves a balanced combination of scalability, performance, and security suitable for distributed campus transaction platforms.

5.2. Practical Implications

The proposed architecture offers several practical benefits for real-world deployment in campus digital ecosystems. Universities and academic institutions can leverage the architecture to develop secure online marketplaces that enable students, faculty, and vendors to exchange goods and services within a trusted institutional environment. The modular microservices design also allows institutions to integrate additional services such as payment systems, campus identity services, and digital resource platforms without disrupting existing system components.

Another important implication lies in the potential integration with university Single Sign-On (SSO) systems. Many institutions rely on centralized identity providers for user authentication across digital platforms. By integrating JWT authentication with SSO identity frameworks, the marketplace platform can provide seamless and secure access for authenticated users while maintaining compatibility with institutional identity management policies.

Furthermore, the architecture can be extended to support multi-campus or federated academic networks. In such environments, multiple institutions may share digital marketplace services while maintaining independent identity and access control policies. The distributed nature of microservices combined with token-based authentication enables scalable cross-campus service interoperability.

5.3. Limitations

Despite its strengths, the proposed architecture presents several limitations that should be considered. First, the system does not incorporate blockchain-based transaction verification mechanisms. Blockchain technologies could provide immutable transaction records and enhanced transparency in marketplace transactions, particularly for financial operations or asset tracking.

Second, the RBAC model implemented in this study relies on static role definitions. While RBAC is efficient and scalable, it may lack flexibility in dynamic environments where access decisions depend on contextual attributes such as time, location, or behavioral patterns. This limitation may restrict the ability of the system to adapt to evolving access control requirements.

Third, the architecture does not incorporate adaptive or risk-based authentication mechanisms. Modern security frameworks increasingly employ dynamic authentication strategies that adjust verification requirements based on user behavior, device trust levels, or anomaly detection models. The absence of such adaptive mechanisms may limit the system's ability to respond to sophisticated attack patterns.

5.4. Recommendations for Future Research

Future research can build upon the findings of this study by exploring more advanced access control and security mechanisms. One potential direction involves the integration of Attribute-Based Access Control (ABAC) with RBAC policies. ABAC extends authorization decisions by incorporating contextual attributes such as user location, time constraints, and resource sensitivity, thereby enabling more flexible and dynamic policy enforcement.

Another promising direction is the integration of zero-trust service mesh architectures. Service mesh technologies such as Istio or Linkerd provide additional security layers through encrypted service-to-service communication, policy enforcement, and traffic monitoring. Incorporating such technologies could further strengthen security within distributed microservices environments.

Formal verification techniques using model checking could also be applied to validate the correctness of authentication and authorization workflows. Formal verification would allow researchers to mathematically prove that access control policies and token validation mechanisms behave as intended under all possible execution conditions.

Finally, future systems may benefit from incorporating artificial intelligence–driven anomaly detection mechanisms capable of identifying suspicious authentication patterns or token misuse. Machine learning algorithms could analyze authentication logs and user behavior to detect abnormal access patterns, thereby

providing an additional security layer capable of identifying emerging threats in real time.

References

1. Dragoni N, Giallorenzo S, Lafuente AL, Mazzara M, Montesi F, Mustafin R, *et al*. Microservices: Yesterday, today, and tomorrow. In: Present and ulterior software engineering. Cham: Springer; 2017. p. 195–216.
2. Ferraiolo D, Sandhu R, Gavrila S, Kuhn D, Chandramouli R. Proposed NIST standard for role-based access control. ACM Trans Inf Syst Secur. 2001;4(3):224–74.
3. Fett D, Küsters R, Schmitz G. The Web SSO standard OpenID Connect: In-depth formal security analysis and security guidelines. In: Proceedings of the IEEE European Symposium on Security and Privacy. 2017. p. 189–204.
4. Fielding RT, Taylor RN. Principled design of the modern Web architecture. ACM Trans Internet Technol. 2002;2(2):115–50.
5. Hardt D. The OAuth 2.0 authorization framework. Internet Engineering Task Force; 2012. (RFC 6749).
6. Hu VC, Ferraiolo D, Kuhn D, Schnitzer A, Sandlin K, Miller R, *et al*. Guide to attribute-based access control (ABAC): Definition and considerations. Gaithersburg (MD): National Institute of Standards and Technology; 2015.
7. International Journal of Scientific Research and Modern Technology. International best reviewer award (July 2025). 2025. Available from: https://www.ijsrmt.com/index.php/ijsrmt/awards/International_Best_Reviewer_Award_july_2025
8. Jones M, Bradley J, Sakimura N. JSON Web Token (JWT). Internet Engineering Task Force; 2015. (RFC 7519).
9. Khan AR, Al-Kahtani MS. Information technology and the transformation of higher education institutions. Technol Forecast Soc Change. 2020;160:120233. (Note: The original entry appears to be incomplete; I have formatted the visible portion.)
10. Li F, Zheng JG. Information technology and the transformation of higher education institutions. Technol Forecast Soc Change. 2020;160:120233.
11. Newman S. Building microservices: Designing fine-grained systems. Sebastopol (CA): O'Reilly Media; 2015.
12. Pautasso C, Zimmermann O, Leymann F. RESTful Web services vs. "big" Web services: Making the right architectural decision. In: Proceedings of the 17th International World Wide Web Conference. 2008. p. 805–14. (Note: Year corrected to 2008 based on standard DOI record for this paper.)
13. Rahman MM, Mahmud I, Hossain MS. Security vulnerabilities in RESTful APIs: A systematic mapping study. J Syst Softw. 2020;169:110708.
14. Richards M, Ford N. Fundamentals of software architecture: An engineering approach. Sebastopol (CA): O'Reilly Media; 2020.
15. Richardson C. Microservices patterns: With examples in Java. Shelter Island (NY): Manning Publications; 2018.
16. Sandhu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. IEEE Computer. 1996;29(2):38–47.
17. Shostack A. Threat modeling: Designing for security. Indianapolis (IN): Wiley; 2014.
18. Taibi D, Lenarduzzi V, Pahl C. Architectural patterns for microservices: A systematic mapping study. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science. 2018. p. 221–32.