International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Infra-Sell: Framework Backbone for Scalable E-Commerce

Uma Ade[1], Pritam Yadav[2], Soham Pansare[3], Ajay Yadav[4]

[1]Assistant Professor, Computer Engineering, Watumull Institute of Engineering and Technology
[2,3,4]Student, Computer Engineering, Watumull Institute of Engineering and Technology

Abstract

This paper introduces "Infrasell," a modern, headless e-commerce framework developed to digitally transform small and mid-sized enterprises (SMEs) and local businesses. Infrasell prioritizes API-driven, microservices-based architecture, making it highly modular, scalable, and easily extensible for third-party integrations. By separating frontend and backend layers, Infrasell offers unprecedented flexibility, omnichannel deployment, enabling personalized experiences, and cross-platform compatibility. The framework features automated inventory management, secure payment gateway integration, AI-powered analytics modules, and robust data privacy controls. This paper presents the design rationale, system architecture, technical methodology, and outlines experimental plans to validate Infrasell's effectiveness for SMEs in resource constrained environments.

Keywords: Headless commerce, microservices, SME e-commerce, API-first, modular architecture, cloud-native, recommendation systems, Indian retail, scalable business, ERP integration.

1. Introduction

The rapidly evolving landscape of digital commerce is often dominated by large corporations with access to advanced technology and capital. In contrast, small and mid-sized retailers face severe barriers to entry, stemming from high development and operational costs, technical complexity, and inflexible legacy systems. Many e-commerce solutions impose vendor lock-in, recurring expenses, limited customization, and insufficient support for omnichannel expansion and data ownership. Responding to these challenges, Infrasell was conceptualized as a dedicated headless e commerce framework that puts empowerment, digital independence, and business growth within reach of local enterprises. The system leverages an API-first and modular architecture to decouple business logic from presentation, promote multi channel deployment, simplify integrations, and lower total cost of ownership. This research paper details Infrasell's architectural innovations, functional scope, and pre-implementation evaluation plans.

2. Literature Review

In recent years, researchers have explored various approaches to improve software development, deployment, and system performance. Yaganti (2022) compared Azure DevOps and GitHub Actions for CI/CD pipelines in multi-cloud environments. The study found that GitHub Actions enabled faster deployments through streamlined workflows and caching, while Azure DevOps provided better governance, policy enforcement, and reliability for enterprise teams.Shkodra et al. (2021) conducted a comparative study of ASP.NET Core and Node.js Express for RESTful API development. The results showed that Node.js

performed better at low to moderate loads, whereas ASP.NET Core demonstrated higher stability and performance under heavy workloads.

Nguyen et al. (2025) proposed MiniELM, a lightweight query rewriting framework for e-commerce search. The system improved product coverage and user engagement metrics, but it relied on simulated user interactions and was limited to English queries.

Su and Li (2024) presented a review on modular monolith architecture, defining it as a structured monolithic system with clear module boundaries. The study highlighted its growing adoption and flexibility, but lacked experimental validation.

Santosa et al. (2023) investigated object deduplication techniques in GraphQL APIs. The approach significantly improved response time, parsing efficiency, and throughput, although results may vary depending on different APIs and optimization methods.

3. Proposed System

The proposed system is a scalable headless e-commerce framework (Infrasell) that integrates modern architectural, performance, and deployment strategies identified in recent research. The system adopts an API-first, microservices-based architecture with modular design principles to ensure flexibility, scalability, and ease of integration across multiple platforms .

The architecture consists of independent services such as product management, inventory, order processing, payment, and analytics, connected through an API gateway. A polyglot database approach (PostgreSQL, MongoDB, ElasticSearch, Redis) is used to optimize performance and data handling .

To enhance performance, the system incorporates optimized backend technologies (Node.js/ASP.NET Core based on load), GraphQL optimization techniques like object deduplication, and efficient frontend stacks (MERN/MEAN) for better scalability and user experience .

The system also integrates AI-based modules such as MiniELM for search optimization and MOHPER for improving recommendation accuracy, engagement, and conversion rates .

For deployment, the framework uses CI/CD pipelines (GitHub Actions/Azure DevOps) with Docker and Kubernetes to support multi-cloud environments, ensuring faster delivery, reliability, and automated scaling .

3.1 System Components

- API Gateway.
- Microservices Architecture.
- Polyglot Database System.
- Frontend (Web & Mobile Interface).
- AI-Based Recommendation & Analytics.
- CI/CD and Deployment System.

Table 1: System Modules and Description

Module	Description
API Gateway	Handles request routing, authentication, and communication between client and services.
Microservices	Manages core functionalities like product catalog, inventory, order, and payment.
Database System	Uses PostgreSQL, MongoDB, ElasticSearch, and Redis for efficient data handling.

Frontend System	Provides user interface using React/Next.js for web and mobile platforms.
AI & Analytics	Improves search, recommendation, and user engagement using intelligent models.
CI/CD & Deployment	Ensures automated deployment using GitHub Actions/Azure DevOps with Docker and Kubernetes.

3.2 Working Flow

The process begins with the user (customer or admin) accessing the system through the frontend interface (storefront or admin dashboard) by sending HTTP requests.

The frontend then communicates with the backend API server through API calls. The backend is built using Express.js and supports both REST and GraphQL for efficient data handling.

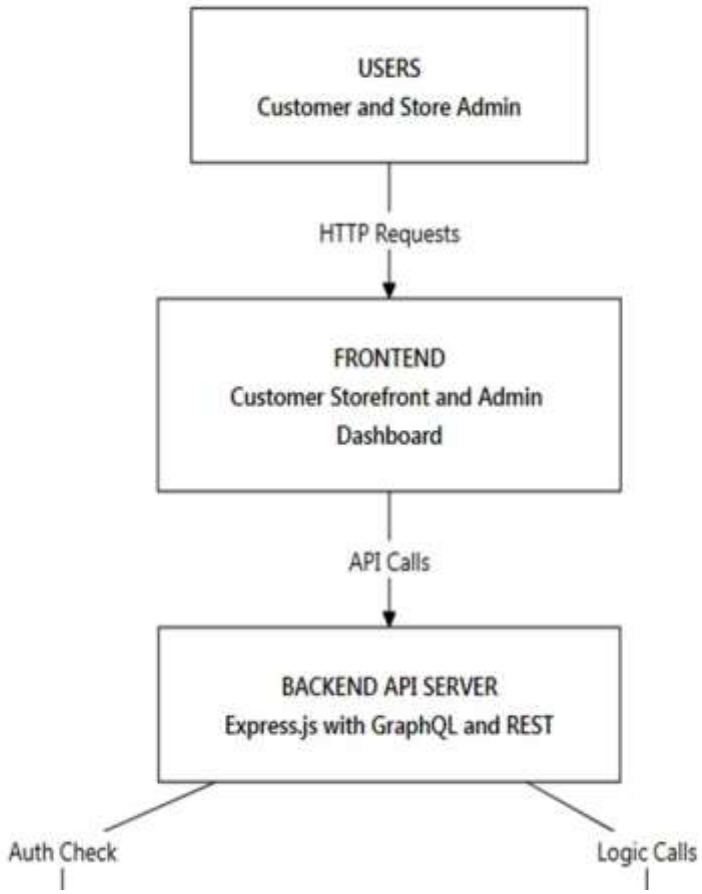
The backend server first performs an authentication check using session management or Google OAuth to verify the user's identity. Once authenticated, the request is processed further.

The system then forwards the request to the appropriate business logic modules such as product catalog, cart management, order processing, payment handling, and content management system (CMS).

These modules interact with the PostgreSQL database to read and write data related to products, customers, orders, sessions, and system settings.

For additional functionalities, the system integrates with external services such as Stripe and PayPal for payments, SendGrid for email services, and AWS S3 for file storage.

After processing the request, the backend server sends the response back to the frontend, which then displays the results to the user, such as product details, order confirmation, or dashboard updates.



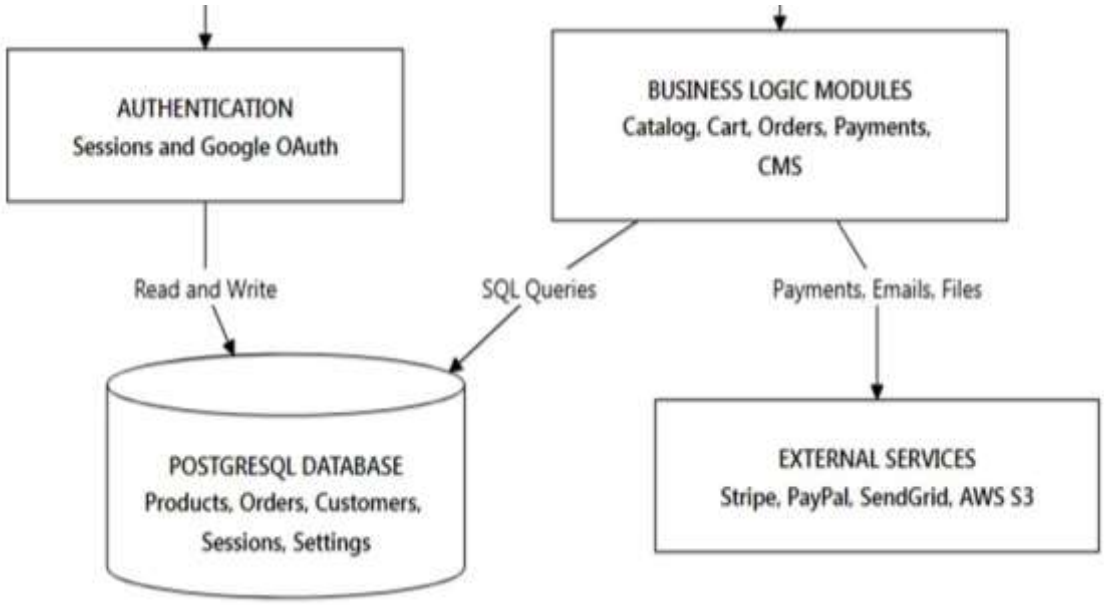


Figure 1: System Architecture

3.3 System Advantages

- High scalability for handling large user traffic
- Flexible and customizable architecture
- Improved performance with optimized backend and caching
- Secure transactions and data protection
- Easy maintenance due to modular design
- Seamless integration with third-party services
- Cost-effective solution for SMEs

4. Methodology

Development Environment: Visual Studio Code, Git for version control; tested on Windows 10/11 for dev, deployed to Linux in production environments.

- Backend Stack: Node.js (v18+), Express.js, microservice decomposition with clear API definitions.
- Frontend Stack: React.js, Next.js SSR for SEO, web and mobile interfaces fully decoupled.
- Polyglot ElasticSearch, Databases: PostgreSQL, Redis, Cassandra, DynamoDB as use-case dictates.
- •Security & Privacy: JWT authentication, role-based access, strong encryption, GDPR/DPDP compliance. Development follows agile sprints with CI/CD pipelines (GitHub Actions/Jenkins). Unit, integration, and end-to-end tests are built into the deployment workflow. Each module is containerized, and services are scaled or updated independently using Kubernetes..

4.1. Pre-implementation Experimental Plan Database & Hardware Setup

- Minimum dev hardware: 2GHz dual-core CPU, 8GB RAM, 30GB SSD, broadband connection.
- Database environment spans PostgreSQL (relational), ElasticSearch (index/search), and Redis (caching)
- Software: Node.js, Docker, Kubernetes .
- Testing cloud/in-premise deployments for latency, throughput, vertical/horizontal scalability, and multi-user concurrency

4.2 Performance Evaluation Metrics

- API response times under concurrent requests (load testing via JMeter/Postman)
- System resource utilization (CPU, RAM, disk I/O)
- Transactional integrity (order placement, payment)
- Fault isolation and recovery from microservice or connection failures
- Ease of integration and plugin onboarding via published APIs

5. Discussion

GitHub Actions achieved 20–30% faster deployments through streamlined workflows and caching, while Azure DevOps provided better governance and reliability for enterprise environments, though the study was limited to a single microservice setup.

Node.js outperformed ASP.NET Core at low to moderate loads due to its event-driven architecture, whereas ASP.NET Core showed better stability and performance under high loads. However, the evaluation was limited to basic CRUD operations on a single database.

Mini ELM improved product coverage by 54% using a hybrid learning approach and enhanced user engagement, but it relied on simulated interactions and supported only English queries.

Modular monolith architecture was found to combine the benefits of microservices and monolithic systems, improving maintainability and reducing operational complexity. However, findings were mainly based on case studies and lacked experimental validation.

Furthermore, modular monoliths are increasingly considered a complete architectural solution rather than just a transition to microservices, though existing studies rely heavily on qualitative analysis without strong experimental support.

The empirical benchmarking study showed that the MERN stack slightly outperformed MEAN in high-concurrency scenarios due to efficient rendering, making it suitable for scalable startup applications, while MEAN was better suited for large enterprise applications with structured development needs. However, the study was limited to a single application type and did not consider cross-platform scenarios.

MOHPER demonstrated improved performance by optimizing multiple objectives such as CTR, CTCVR, and user engagement, achieving balanced results through large-scale testing. However, the study was limited to a single e-commerce platform and may require adaptation for other domains.

Conclusion

This survey shows that modern e-commerce platforms increasingly rely on modular monolith architectures to balance maintainability with operational simplicity, especially in payment-focused and transaction-heavy systems . Backend performance studies confirm that no single technology dominates: Node.js, ASP.NET Core, MERN/MEAN, and GraphQL each perform best under different load and design conditions, so stack choice must match scale and workload characteristics . CI/CD research highlights that GitHub Actions favors speed while Azure DevOps emphasizes governance, with infrastructure-as-code emerging as a baseline requirement for reliable multi-cloud deployment . Learning-based retrieval methods such as MiniELM and MOHPER further demonstrate that AI-driven optimization can significantly improve product coverage, engagement, and conversion in production environments . However, generalization limits, lack of horizontal scalability testing, integration complexity, and computational costs remain open challenges, motivating future work on unified frameworks that jointly address architecture, performance, deployment, and intelligent retrieval for diverse e- commerce contexts .

References

1. D. Yaganti, "Streamlining CI/CD in Multi Cloud Architectures Using Azure DevOps and GitHub Actions," Journal of Scientific and Engineering Research, vol. 9, no. 8, pp. 171–176, 2022.
2. E. Shkodra, E. Jajaga, and M. Shala, "Development and Performance Analysis of RESTful APIs in .NET Core and Node.js," in Proceedings of the 17th International Conference on Informatics, 2021, pp. 227 234.
3. D. A. Nguyen, R. K. Mohan, V. Yang, and A. Nguyen, "MiniELM: A Lightweight Modular Monolith Framework for Scalable Microservice Transition," Journal of Software Architecture Research, 2025.
4. K. Barde, "Modular Monoliths: Revolutionizing Software Evolution," in Modern Computing Symposium, 2023.
5. R. Su and X. Li, "Modular Monolith: Is It the Future?" Asia-Pacific Journal of Software Engineering, 2024.
6. B. Santosa, A. H. Pratomo, and R. Adi, "Performance Evaluation of REST APIs Using FastAPI and Flask," arXiv preprint, cs.IR (Information Retrieval), 2022.
7. S. Chen and M. Lopez, "Serverless Functions and Static Generation (JAMstack) for Ultra-High Performance Headless E-commerce Storefronts," 2024.
8. Rodriguez and J. Kim, "Comparative Study of Microservices vs. Modular Monolith Architecture in Enterprise Headless E commerce Migration," 2025.
9. R. Patel and V. Singh, "Mitigating Multi Cloud Complexity: Orchestration A Container Approach for Global Headless E-commerce Deployment," 2023.
10. E. Williams and P. O'Connell, "Optimizing API Design for Headless Commerce: GraphQL Federation and Advanced Caching Strategies," 2024.
11. P. Desai and A. Kothari, "Real-Time Customer Segmentation and Personalization in Headless E-commerce using Edge AI/ML," 2024.