National Conference on Information Technology & Digital Application (NCITSA-2025)
International Journal of Advanced Engineering, Management and Science (IJAEMS) Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.58 ISSN: 2454-1311

Role-Based Architecture and Development

Ms. Smita Agrawal[1] , Bhumit Jain[2], Bhuvnesh Saini[3]
[1]Professor Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur
[2]Student Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur
Email: bhumitjain.it25@gmail.com
[3]Student Department of Information Technology, Jaipur Engineering College & Research Centre, Jaipur
Email: bhuvneshsaini.it25@gmail.com

***Abstract*—**A modular, scalable, and domain-agnostic service management platform architecture is proposed to streamline interactions between service providers and consumers across domains such as home maintenance, IT consulting, education, and healthcare. The system is built using a microservices architecture with Golang, complemented by a dynamic frontend developed in Angular and a DynamoDB database employing a schema-per-domain model for data isolation. Real-time communication is enabled via REST, ensuring low-latency interactions. Secure access is achieved through JWT-based authentication. The entire system is containerized using Docker and orchestrated through Kubernetes to support scalability, service discovery, and fault tolerance. CI/CD pipelines are implemented for continuous testing and deployment. The design adheres to IT Service Management (ITSM) principles and incorporates the PLASMA framework for event-driven responsiveness. Clean Architecture and SOLID principles guide the overall structure to ensure maintainability and long-term extensibility. This integration of modern software paradigms delivers a robust foundation for scalable, resilient, and adaptive service-based systems.

Keywords- Service Management, Microservices, Golang, Angular, PostgreSQL, WebSockets, JWT, Kubernetes, Docker, ITSM, , Clean Architecture, CI/CD

I.INTRODUCTION

The digital transformation of service delivery has advanced rapidly in recent years, driven by evolving consumer expectations and technological innovation. Modern users increasingly demand seamless, real-time interactions and highly personalized experiences across diverse service sectors such as healthcare, education, IT support, and home maintenance. In contrast, traditional monolithic service platforms—once widely adopted across enterprises—struggle to meet these demands due to limitations in scalability, adaptability, and maintainability. Their tightly coupled structures hinder modular upgrades and rapid deployment cycles, making them ill-suited for today's dynamic service environments.

ServiceNest emerges as a response to these limitations, offering a modern, modular, and domain-agnostic solution built on a microservices foundation. By decomposing functionality into independent and loosely coupled services, ServiceNest supports modular development, autonomous scaling, and domain-specific customization. This architecture enables the platform to serve a wide range of service verticals—such as appointment scheduling, remote consultations, and on-demand bookings—within a single, cohesive ecosystem.

Central to ServiceNest's design is the application of Domain-Driven Design (DDD). Each domain encapsulates its business logic within well-defined bounded contexts, implemented as isolated microservices. These services manage their own database schemas and expose independent APIs, ensuring minimal interdependency and improved scalability.

International Journal of Advanced Engineering, Management and Science (IJAEMS) Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.58 ISSN: 2454-1311

To meet the need for responsive user experiences, ServiceNest incorporates WebSocket-based real-time communication channels that ensure low-latency updates and enhance operational efficiency. The frontend, developed using Angular 16+, includes lazy-loaded modules, dynamic dashboards, and role-specific views tailored to the distinct needs of consumers, service providers, and administrators.

The backend is implemented in Golang, following the principles of Clean Architecture, which separates domain logic from infrastructure and external dependencies. This layered design improves testability, flexibility, and maintainability. Furthermore, the platform is containerized using Docker and orchestrated via Kubernetes, enabling high availability, fault tolerance, and elastic scaling. Integrated CI/CD pipelines ensure seamless deployment and version control across development and production environments.

This paper introduces the architectural foundations, design decisions, and technical implementations of ServiceNest. It examines how theoretical constructs like microservices, DDD, and Clean Architecture are applied in practice to create a scalable and resilient platform for service management in diverse domains.

II. Background Study

The evolution of service-oriented platforms has been driven by the need for modular, scalable, and maintainable systems capable of adapting to diverse domain-specific requirements. Legacy monolithic architectures, although once dominant in enterprise environments, present significant challenges such as tight coupling, limited scalability, and inflexibility in deployment. These limitations hinder responsiveness to evolving user needs and slow down innovation cycles.

To address these issues, modern service platforms leverage **microservices architecture**, decomposing system functionality into loosely coupled, independently deployable services. This promotes domain-specific customization, seamless updates, and horizontal scalability. Complementing this architectural choice is **Domain-Driven Design (DDD)**, which structures applications around business domains with well-defined boundaries and minimal cross-service dependencies.

Robust backend development is achieved using **Golang**, a statically typed, compiled language known for its performance and concurrency support. Clean code practices are enforced through **Clean Architecture** and **SOLID principles**, ensuring maintainability, scalability, and clear separation between domain logic and infrastructural concerns.

RESTful APIs serve as the primary communication mechanism between microservices and the frontend, enabling stateless, scalable, and easily debuggable interactions. These APIs allow clients to perform CRUD operations and retrieve domain-specific data in a consistent and predictable manner. API security is enforced using **JWT-based authentication** and **Role-Based Access Control (RBAC)**, providing secure and fine-grained access management.

The frontend is built using **Angular**, leveraging modular architecture, reactive programming with **RxJS**, and a component-based UI structure. Lazy loading and role-specific views enhance performance and user experience. State management is achieved through services and observables that interact with the REST APIs.

Operational resilience and portability are ensured through **Docker-based containerization** and **Kubernetes orchestration**, enabling elastic scaling, fault tolerance, and service discovery. Deployment pipelines are streamlined using **CI/CD** practices powered by tools like GitHub Actions, supporting automated builds, testing, and rollout.

Observability is enhanced with **Prometheus** for metrics collection and **Grafana** for visualization, while structured logging is centralized using **Fluentd**. These monitoring and logging tools support proactive issue detection and system optimization.

III. Proposed work

The proposed work focuses on the design and development of a robust, scalable, and domain-agnostic service management architecture that leverages microservices and modern software engineering paradigms to meet the evolving needs of service-oriented applications. The core objective is to establish a flexible and secure framework capable of supporting multiple service verticals through modular components and stateless communication.

At the heart of this architecture lies a **microservices-based backend**, developed using **Golang**, which adheres to **Clean Architecture** principles. This ensures separation of concerns by decoupling domain logic from infrastructural dependencies

International Journal of Advanced Engineering, Management and Science (IJAEMS) Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.58 ISSN: 2454-1311

and delivery mechanisms. The backend is composed of loosely coupled services that expose **RESTful APIs**, allowing for consistent and scalable interactions with client-side applications.

Each microservice operates within a bounded context, aligned with **Domain-Driven Design (DDD)** principles, and manages its own **PostgreSQL schema** to ensure data isolation and integrity. The use of **Flyway** for schema versioning enables zero-downtime migrations and maintains data consistency across deployments.

On the frontend, the application is built using **Angular 16+**, leveraging lazy-loaded modules and reactive programming with **RxJS** for optimal performance. The user interface is designed with accessibility in mind, using **Angular Material** and adhering to **ARIA standards**. Role-specific views and route guards ensure tailored access to different types of users such as administrators, service providers, and consumers.

Security is a critical aspect of the proposed system. **JWT (JSON Web Token)** authentication is employed to enable stateless, secure API access, while **Role-Based Access Control (RBAC)** enforces fine-grained permissions at the service level. Middleware components handle request validation, rate limiting, and structured logging.

The system is fully containerized using **Docker**, with services deployed and orchestrated via **Kubernetes**, enabling high availability, scalability, and resilience. **CI/CD pipelines** automate the build, test, and deployment process, ensuring rapid and reliable delivery of updates. Configuration and secrets management are handled using **Helm charts** and **Vault**, enhancing operational security and consistency.

The architecture is designed with observability in mind. **Prometheus** and **Grafana** are integrated to monitor system metrics and visualize performance trends, while **Fluentd** captures structured logs for efficient debugging and auditability.

By combining these technologies and best practices, the proposed work aims to deliver a general-purpose service management framework that is adaptable across domains, resilient under real-world conditions, and maintainable over time. Future enhancements will focus on integrating AI-driven automation, predictive analytics, and multilingual support for broader applicability.

IV.CONCLUSIONand Future Scope

This research outlines the development of a scalable, modular, and domain-agnostic architecture for service management systems using state-of-the-art software engineering practices. By integrating a microservices-based backend in **Golang**, a dynamic frontend using **Angular**, and a schema-per-domain data strategy with **PostgreSQL**, the proposed architecture addresses key challenges such as system scalability, maintainability, and adaptability across various service sectors.

The application of **Clean Architecture** and **SOLID principles** ensures clear separation of concerns and long-term extensibility, while **RESTful APIs** offer a stateless and standardized communication mechanism between services and clients. Security is reinforced through **JWT-based authentication** and **RBAC**, enabling secure and role-specific access to resources.

Deployment resilience and portability are achieved through **Docker containerization** and **Kubernetes orchestration**, ensuring high availability, elastic scaling, and fault tolerance. With integrated **CI/CD pipelines**, the system supports rapid development and release cycles. Observability is enhanced through monitoring and logging tools such as **Prometheus**, **Grafana**, and **Fluentd**, promoting operational transparency and optimization.

Future Scope:

To further evolve the system's capabilities and extend its applicability, several future directions are envisioned:

- **AI/ML Integration:** The use of machine learning models for intelligent routing, predictive analytics, and automated service recommendations will enhance system intelligence and responsiveness.
- **Blockchain Integration:** Incorporating blockchain technologies can offer immutable records for transactions, identity verification, and audit trails, enhancing transparency and trust.

International Journal of Advanced Engineering, Management and Science (IJAEMS) Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.58 ISSN: 2454-1311

- **Native Mobile Applications:** Building cross-platform mobile applications using frameworks like **Flutter** or **Kotlin Multiplatform** will extend accessibility and usability for end-users.
- **Internationalization (i18n):** Adding multilingual and locale-aware support will prepare the system for global deployment and cater to diverse user bases.
- **Edge Computing & Offline Support:** Implementing edge capabilities and Progressive Web App (PWA) features will enable offline operations and reduce latency in remote environments.

Through these enhancements, the proposed architecture aims to remain future-ready, scalable, and applicable across an expanding array of real-world service scenarios.

ACKNOWLEDGEMENTS

We extend our sincere thanks to Jaipur Engineering College and Research Centre for providing the infrastructure and environment to pursue this research.

REFERENCES

1. Maddevs.io. (n.d.). *Creating Backend Service with Go Clean Architecture*. Retrieved from https://maddevs.io/blog/creating-backend-service-go-clean-architecture/
2. Selvaganesh. (n.d.). *Angular Scalable Project Structure*. Medium. Retrieved from https://medium.com/@selvaganesh93/angular-scalable-project-structure-8806f8f1405a
3. MDPI. (2021). *The Evolution of ITSM in the Modern Era*. *Future Internet*, 13(9). Retrieved from https://www.mdpi.com/1999-5903/13/9/237
4. Martin, R. C. (n.d.). *SOLID Principles of Object-Oriented Design*. Retrieved from https://web.archive.org/web/20200810133253/http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

International Journal of Advanced Engineering, Management and Science (IJAEMS) Special Issue
https://dx.doi.org/10.22161/ijaems.ncitsa.2025.58 ISSN: 2454-1311