Available Online at http://www.journalajst.com

ASIAN JOURNAL OF SCIENCE AND TECHNOLOGY

ISSN: 0976-3376

Asian Journal of Science and Technology
Vol. 16, Issue, 10, pp. 13918-13922, October, 2025

RESEARCH ARTICLE

DESIGN OF A MICROSERVICES ARCHITECTURE IN VIRTUAL CONTAINERS

Samantha Yazmin Elizalde Valencia*, José Juan Hernández Mora, María Guadalupe Medina Barrera and Juan Ramos Ramos

Instituto Tecnológico de Apizaco

ARTICLE INFO

Article History:
Received 11th July, 2025
Received in revised form
19th August, 2025
Accepted 24th September, 2025
Published online 27th October, 2025

Keywords:
Architecture, Containerization, Docker, Microservices.

Corresponding author:
Samantha Yazmin Elizalde Valencia

ABSTRACT

This article presents the methodological proposal for the design and implementation of a containerized microservices architecture. The proposal covers the methodologies used for the research phase of the initial stages of project development, as well as the methodology used to create a test computing system. An architecture based on domain architectures is proposed, as well as the design of the system's physical architecture, based on the requirements presented by the collaborating institution for the test case. The results describe the configuration of the microservices and containers, as well as their integration into a common network of running services.

Citation: ***Samantha Yazmin Elizalde Valencia, José Juan Hernández Mora, María Guadalupe Medina Barrera and Juan Ramos Ramos.*** **2025.** "New field concept by Superposition", *Asian Journal of Science and Technology*, 16, (10), 13918-13922.

INTRODUCTION

Currently, countless organizations remain trapped with software built on obsolete and fragile foundations; the magnitude of this problem is quite extensive, as approximately 65% of enterprise applications are legacy systems. The most frustrating thing is seeing how these organizations waste 60% to 80% of their IT budgets simply maintaining these completely obsolete systems, instead of investing in innovations that could truly transform their businesses (Occena, 2025). These legacy systems work, but with many difficulties, built on monolithic architectures that are hard to manage, with poor or nonexistent documentation, and dependent on technologies that are often already obsolete; integrating them with modern platforms is an even more complex task (Sánchez, 2024). Microservices architectures emerge in this landscape as an encouraging alternative. Studies suggest that implementing microservices in containers can improve performance by up to 15%, since each component occupies its own space and is deployed almost instantly (Saransig, 2018). This article presents the proposed methodologies for designing microservice-based architectures using Docker containers.

Proposed methodologies for project development

This section presents the suggested methodologies for developing the project, from the information study and analysis phase to the phase detailing how the software system is developed.

RESEARCH METHODOLOGY

After analyzing several research methodologies, two approaches were selected that met certain important characteristics for carrying out the project: technological research and technological research in engineering. These methodologies are suitable for projects where the primary objective is not to establish new theories but to reconstruct processes by adapting and improving existing solutions. Technological research will allow us to examine existing solutions, take their best elements, and adapt them to the project's needs (Bello, Retrieved March 6, 2025).Technological research in engineering adds a component of utmost importance: the intervention project (De la Cruz Casaño, 2016). As can be seen in Figure 1, this component goes beyond theoretical analysis, requiring the implementation of a solution that can be evaluated. In this case, this translates into the development of a functional system based on the proposed architecture. This will allow us to verify whether the requirements outlined in the research question are met. Additionally, it incorporates an iterative process that allows for constant feedback during the various stages of the project.

The following stages are proposed for conducting the research:

1. Problem identification: This stage focuses on identifying and formulating the research problem; it is here that specific objectives are established and the scope of the research is defined.

2. **Literature review:** Conduct a review of the literature related to the research topic, identifying background and methodologies used in other research studies.
3. **Information analysis:** In this phase, some solutions to the problem are developed or new questions about the topic are raised, using the information gathered in the previous phase.
4. **Project development:** Development of the intervention project: In this section, the system is developed and implemented using a software development methodology.
5. **Validation and verification:** This stage is carried out by applying test cases.
6. **Results:** Interpret the results obtained in light of the research objectives, and identify possible areas for improvement.
7. **Conclusions:** Draw conclusions based on the findings obtained, and suggest recommendations for future research.

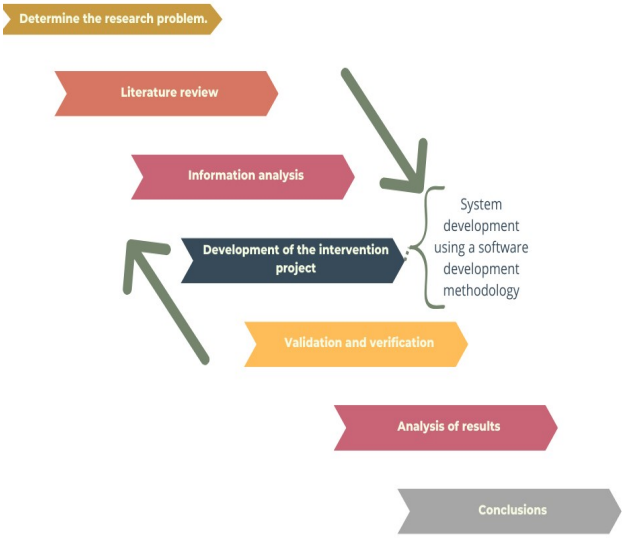

Figure 1. Research methodology

Computer System Development Methodology: For project management, an agile methodology is proposed that merges elements of Scrum and Kanban. As shown in Figure 2, Scrum provides the necessary structure through sprints with deliverables and periodic reviews that allow for adjusting the project's course (Estrada, 2021). Kanban provides a visual framework through its boards, which allows for improved sprint planning (Castro, 2025). Given that the development team is made up of a small group (three members), the Small-Scale Scrum variant (Garcia, 2019) is implemented, suitable for promoting self-organization, constant communication, and shared responsibilities in compact teams. This methodological combination allows for better iteration planning with continuous feedback. The methodology will be segmented into Sprints with the goal of maintaining a pace of progressive deliveries and adapting to client demands.

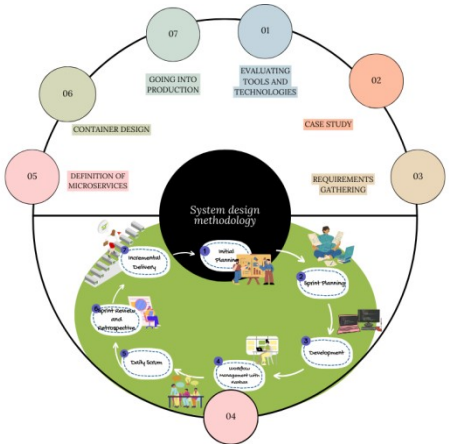

Figure 2. System development methodology

1. **Tool and Technology Evaluation:** Compare container tools and APIs.
2. **Case Studies:** Analyze case studies with similar attributes to identify best practices and challenges.
3. **Requirements Gathering:** In this stage, stakeholder interviews will be conducted to identify system needs.
4. **Sprint Planning:**
   - Initial Planning. The next stage aims to establish the general scope of the project, objectives, roles, and the initial backlog.
   - Sprint Planning. The work to be carried out during the sprint is organized.
   - Workflow Management using Kanban. This stage ensures a steady flow of work and eliminates bottlenecks.
   - Daily Scrum. The purpose of this stage is to monitor progress and quickly resolve blockers.
   - Sprint Review and Retrospective. The functional progress is shown to the Product Owner and feedback is received.
   - Incremental Delivery. At the end of each sprint, deliver product versions.
   - Repeat the Cycle. Explore and incorporate improvements or new features through continuous iterations.
5. Microservice Definition. In this segment, the microservices that will run in the system will be selected.
6. **Container Design:**
   Develop the structure that will incorporate each container.
7. **Production Deployment:**
   Implement the architecture in the production environment.

System Architecture

This section presents the suggested architectural design for the system, detailing both its layered logical structure and its physical implementation. The architecture establishes the modular structure of the system, detailing the different functional layers and their interrelationships, while the physical architecture illustrates the precise layout of the infrastructure elements, specifically tailored to the technical and operational needs of the collaborating institution.

Logical view of the system

The software architecture shown is based on domain-driven design principles, combined with a microservices perspective (see Figure 3). The core of this perspective is based on a detailed understanding of the business domain, which requires close collaboration between developers and domain experts (Osorio, 2023). This structure separates the core logic (kernel) from the technological components thru layers, while incorporating microservices for specific functional elements. This significantly simplifies system maintenance and evolution (López, 2017). This architecture provides a flexible, scalable, and modular working environment, specifically designed to support the continuous evolution of complex systems and their potential migration to cloud environments. The solution implements a layered structure of independent layers that isolates the core business rules from external dependencies (frameworks, databases, and interfaces) (Martin, 2017).

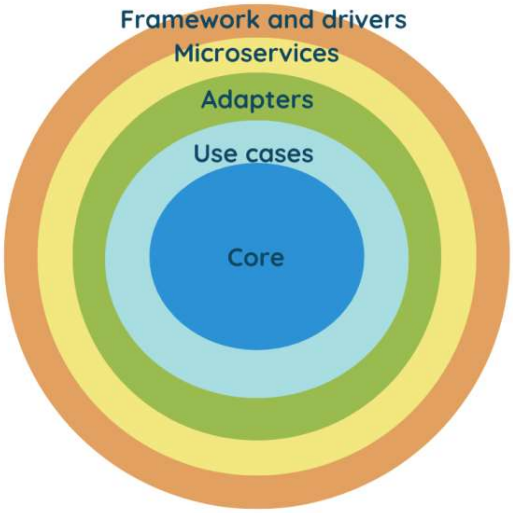

Figure 3. System development methodology

- Core (Business Logic): The core houses the entities and business thinking, based on the principle of onion and clean architecture. This ensures that business logic is separated and autonomous from the infrastructure.
- Use Cases: These implement specific use cases and communicate with the core through interfaces. This is inspired by the hexagonal architecture, in which services act as mediators.
- Adapters (Interfaces): Ports and adapters link the core and services to external systems such as databases, external APIs, and others. This ensures that the core remains autonomous from the details of infrastructure implementation.
- Microservices: Each feature of the system is implemented as a self-contained microservice, deployed in containers. This enables scalability and continuous deployment, in accordance with microservices architecture best practices.
- Frameworks and Drivers: Considering the clean architecture, this layer is located externally, where it can cause minimal damage. It usually includes frameworks and tools such as database and web framework.

Physical view of the System: To test the proposed methodology, a sports management system will be developed. This system will be used by an institution whose strategic objective is to provide medical care to athletes seeking to improve their physical performance in specific representative competitions. Through this system, both functional and non-functional requirements were identified, focusing on: managing sports medical appointments, managing athletes' medical records, generating statistical and monitoring reports, and fully controlling the medical consultation cycle. This analysis enabled the creation of the following essential artifacts: use case diagrams, abuse cases, class diagrams for sports medical entities, clinical and administrative process flows, and behavioral specifications for key modules. The findings of this study established the basis for the physical design of the system, as shown in Figure 4, ensuring that the suggested architecture meets the identified needs and established quality criteria.

Within the architecture, users can access the application from browsers on computers, tablets, and phones, communicating via HTTP requests. Based on these requests, the frontend, developed in React, offers a visual interface. React manages system styles through CSS files and uses JavaScript to enhance interactivity and the user experience. When a user interacts with the interface, the frontend sends a request to Django, which in this case acts as middleware through the Django REST Framework. The latter acts as a link between the visual layer and the microservices. In the backend, data models play a crucial role, representing the structure of the information and its storage in the database.

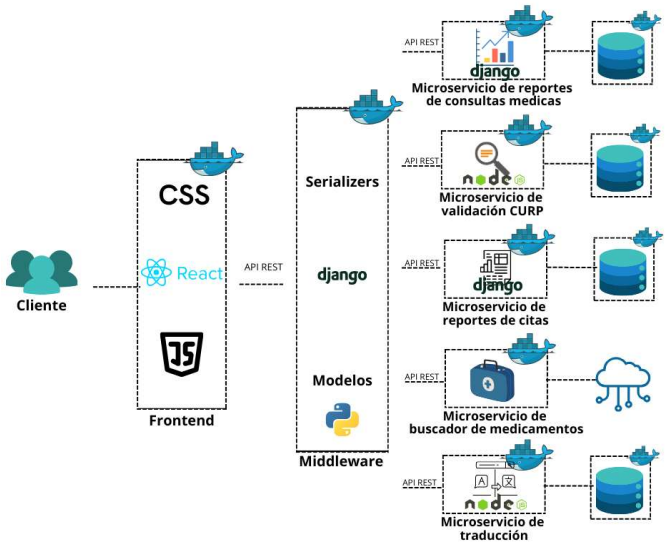

Figure 4. Physical view of the system

The system has several microservices created to improve and verify information on forms.

The system incorporates the following microservices:

Credential Validation Microservice: Validation of athletes' CURPs using a server-hosted Node.js service. The implementation will be based on a cloned Git repository.

Analytics and Reporting Microservices: Generation of statistical charts, creation of customized PDF reports with user-adjustable parameters, and secure access to the database via ORM.

Medication Finder Microservice: This microservice is intended to be integrated into the medical consultation section. It enables healthcare professionals to make better decisions regarding medications and provides information on recommended doses, adverse reactions, and other relevant data. This microservice belongs to the U.S. Department of Health and Human Services and is hosted in the cloud and will be accessed through an API. It also requires an additional service to operate.

Translation microservice: Since all the information in the drug search engine was available in English, a translator was needed to display the information in the appropriate language for the context of use. This microservice belongs to lingva-translate. It allows translation for both the request and the information received, using a Node.js service. To ensure scalability and portability, each essential element of the system is hosted in Docker containers, allowing deployment on both local and cloud servers, depending on the client's requirements. Interaction between various containers and microservices is achieved through REST APIs.

Frontend and backend configuration: The system aims to operate in a modular fashion, allowing for the addition of new features without significantly compromising system performance; this is why it is necessary to separate the frontend and backend functionalities. To develop the frontend, we used the React development framework, hosted on port 5173, which handles everything related to the visual aspects of the system. The backend was based on the Django development framework, which enabled the creation of all data models and the management of the application's core logic. This service runs on port 8000. To achieve communication between both parties, we used the Django Rest Framework, which enabled communication by converting the data models to JSON format through views and serialization files.

Microservice Configuration: Regarding the services created by the development team, the configuration is very similar, as these two

microservices for generating reports and statistics on appointments and consultations are located in two different Django projects, hosted on ports 8001 and 8002. To communicate, information is obtained from the URLs belonging to the backend microservice. This allows these microservices to process and filter the relevant data for presentation on the frontend, either through graphs or to enrich the information contained in the generated PDF files. While the microservices cloned from Git repositories correspond to the CURP translation and validation service, they are developed in Node.js and are hosted on ports 3000 and 3001. For their execution, it was necessary to install the requirements specified in their documentation. Their information is managed by consuming their respective APIs. The APIs will be responsible for managing the requests necessary for the frontend to communicate with one of the two external services. In the case of the external microservice hosted in the cloud, a function is defined that queries the FDA API to obtain drug information, performs a search for information, and returns a list of processed objects.

Container Configuration (Docker): To containerize the microservices, it was necessary to create a Dockerfile within the corresponding folder for each one. This allowed the creation of the image for each microservice, specifying the base environment on which the image will be built, the working directory, the installation of dependencies, the exposure of dependencies, and the startup command. The Docker Compose file also allowed the management of multiple containers, assigning a network over which they will communicate. This option is especially useful for applications with multiple services.

RESULTS AND DISCUSSION

This section presents the results achieved during the project execution. For this reason, the results obtained from the application evaluation are presented using test cases for each of the microservices. Performance was evaluated using load and stress tests performed using the Locust tool.

Test Case Results Analysis: The test cases evaluated the performance of five microservices: CURP validation, medical appointment reporting, medical consultation statistics, medication search, and medical information translation. The findings generally indicate a high degree of efficiency and accuracy in the responses to the microservices, with the majority of cases marked as "Passed." This suggests that the systems meet functional requirements and adequately handle error cases.

However, the reason for the inability to complete the process has not been specified. This indicates a potential area for improvement in the consistency of the error messages displayed by the system. The results were favorable for the reporting and statistics microservices (VAL_002 and VAL_003). The filters applied (for the athlete, area, dates, healthcare professional, etc.) fields worked appropriately, producing segmented reports and accurate statistics. Furthermore, the system correctly handled cases where no data was available, displaying clear messages to the user. This demonstrates an appropriate design for query management and data presentation. In the drug search test (VAL_004), the microservice effectively responded to valid searches, providing specific data such as dosages and adverse reactions. It also appropriately handled cases of invalid names or missing parameters, providing clear error messages. This demonstrates robust request and error management. Finally, the translation microservice (VAL_005) demonstrates its ability to convert medical data into the required language and handle circumstances such as empty text or unsupported languages. Furthermore, in connection interruption tests, the system demonstrated its resilience by remaining stable and providing information messages. These findings corroborate the reliability and effectiveness of the systems in a test environment, establishing a backup for their implementation in the production process.

Load Test Analysis: Load tests performed with Locust (200 and 100 users) show behavior relevant to the system architecture: the microservices operate autonomously, and although some bottlenecks exist in a microservice, they do not reduce the overall performance of the system. This reinforces the benefit of a decoupled architecture, in which the modification or optimization of one service does not impact the others.

Key Findings

System Resilience: Although Django 8001 (Citation Statistics) showed high latencies, the other services (Node, React, Django 8000/8002) maintained optimal performance. This demonstrates that the system is well decoupled, so one slow service doesn't drag down the others.

Overall Throughput: Remained stable, indicating that the current infrastructure scales appropriately.

Different Tools: Node.js and React stood out for their efficiency, ideal for critical services such as CURP validation or user interfaces, while Django (especially in medical consultations) showed limitations in handling high loads, but its flexibility allows for specific improvements (e.g., caching, query optimization).

Figure 5. Results of the applied test cases

Microservice	Test case ID	Compliance	Recommendation
CURP Validation	(VAL_001)	Correct validation of formats and data.	Check consistency between error messages and blocking actions.
Appointmentreports	(VAL_002)	Filters (area, dates, athlete) work correctly.	-
Medical consultations	(VAL_003)	Segmented reports and precise date management.	-
Drugfinder	(VAL_004)	Detailed responses for valid searches. Clear error handling.	Improve messages for empty searches.
Translation	(VAL_005)	Accurate translation and handling of external errors.	-

Figure 6. Load test analysis

Service	Latency	Throughput (RPS)	% Failures	Observation
Node 3000 (CURP)	5 ms (200u) / 2 ms (100u)	29.7 / 18.9	0 %	Excellent performance, minimal latency even under high load.
React 5173 (Frontend)	5 ms (200u) / 2 ms (100u)	29.7 / 18.9	0%	Quick responses, ideal for user experience.
Django 8000 (Middleware)	42 ms (200u) / 25 ms (100u)	30.7 / 18.9	0%	Stable, acceptable latency for both loads.
Django 8001 (Citation Statistics)	400 ms (200u) / 200 ms (100u)	91.2 / 19.6	0.06%	Bottleneck: Latency spikes (>5s at 200u). Requires optimization (SQL/cache).
Django 8002 (Query Statistics)	190 ms (200u) / 23 ms (100u)	32.1 / 19.6	0%	Good performance in 100-user test, but latency increases with more load.

During CURP validation (VAL_001), the microservice demonstrated the ability to adequately distinguish between valid and invalid formats and identify incorrect data or structures. However, unexpected behavior was detected in the handling of a CURP with an invalid format, in which the system did not allow the process to continue.

Improvement Opportunities: The Appointment Statistics service (Django 8001) is the only one that urgently requires optimization. Since the rest of the system does not depend on its speed, modifications can be implemented without risk of affecting other modules.

Figure 7. Total request per secong

CONCLUSIONS

The proposed architecture based on containerized microservices (Docker) proves to be a replacement for legacy systems, offering scalability, modularity, and portability. This article presented the methodology, which ranges from analysis to production, validating the architecture through a real-world case study. It was observed that containerization simplifies the deployment and management of microservices, reducing dependency conflicts. The combination of Django (backend) and React (frontend) allows for a separation of responsibilities, while using the Django REST Framework facilitated communication with external services. The microservices validated through test cases, which included circumstances with different inputs, both incorrect and correct, demonstrated robust and resilient operation. Despite partial outages, the core system maintained its operation, restricting the impact only to the impacted services, strengthening fault resilience and operational continuity. This behavior is confirmed in load tests, where services like Node.js and React maintained excellent performance even when others (like Django 8001) showed high latencies, demonstrating that decoupling prevents fault propagation and enables selective optimizations without impacting the overall system.

REFERENCES

Bello, F. (s. f.). Reflexión: La investigación tecnológica o cuando la solución es el problema. *Revista de la Facultad de Ingeniería, Universidad de Carabobo*. http://servicio.bc.uc.edu.ve/faces/revista/a6n13/6-13-3.pdf

Castro, A. (2025, 13 de enero). *Scrum y Kanban: Una combinación efectiva para la gestión ágil*. LinkedIn. https://www.linkedin.com/pulse/scrum-y-kanban-una-combinaci%C3%B3n-efectiva-para-la-gesti%C3%B3n-aitor-castro-sulnf/

De la Cruz Casaño, C. 2016. *Metodología de la investigación tecnológica en ingeniería*. Universidad Continental. https://www.academia.edu/94930372/Metodología_de_la_investigación_tecnológica_en_ingeniería

Estrada-Velasco, M. V., Saltos-Chávez, P. R., Núñez-Villacis, J. A., & Cunuhay-Cuchipe, W. C. (2021). Revisión sistemática de la metodología Scrum para el desarrollo de software. *Revista Científica*. https://dialnet.unirioja.es/servlet/articulo?codigo=8384028

Garcia, D. 2019, 19 de febrero. *Scrum para equipos pequeños, una introducción*. IntelDig. https://www.inteldig.com/2019/02/scrum-equipos-pequenos/

López Hinojosa, J. D. 2017. *Arquitectura de software basada en microservicios* [Tesis de grado, Universidad Técnica del Norte]. 1Library. https://1library.co/document/y81dmvwz-arquitectura-software-basada-microservicios-desarrollo-aplicaciones-asamblea-nacional.html

Martin, R. C. 2017. *Clean architecture: A craftsman's guide to software structure and design*. Prentice Hall.

Occena, R. A. 2025, 21 de marzo. *The cost of legacy systems: How outdated IT holds companies back*. LinkedIn. https://www.linkedin.com/pulse/cost-legacy-systems-how-outdated-holds-companies-back-andre-occec

Osorio Tibán, G. E., & Campos Sánchez, L. F. (2023). *Desarrollo de un prototipo de microservicios con Clean Architecture* [Tesis de grado, Universidad Politécnica Salesiana]. Repositorio UPS. https://dspace.ups.edu.ec/bitstream/123456789/28179/1/MSQ862.pdf

Sánchez, L. 2024. *Sistemas Legacy - Modernizando la infraestructura tecnológica*. Initium. https://www.initiumsoft.com/blog_initium/sistemas-legacy/

Saransig Chiza, A. F. 2018. *Análisis de rendimiento entre una arquitectura monolítica y una arquitectura de microservicios – tecnología basada en contenedores*. Repositorio CORE. https://core.ac.uk/download/pdf/200323828.pdf

****