

Infraestructura local para espacios inteligentes: uso de docker y microservicios en el monitoreo de adultos mayores

**Cesar Abraham Delgado Cardona, Carlos Lino Ramírez, Víctor Manuel Zamudio
Rodríguez, David Asael Gutiérrez Hernández, Rafael Santos Pérez**

**Instituto Tecnológico de León
León, Guanajuato, México**

Resumen

La automatización de entornos inteligentes mejora la seguridad y calidad de vida de personas vulnerables, como los adultos mayores. Sin embargo, muchas soluciones existentes dependen de la nube, lo que incrementa los costos y compromete la privacidad y la latencia. Este trabajo propone una arquitectura descentralizada basada en microservicios y contenedores Docker, diseñada para operar de forma local utilizando hardware de bajo costo como la Raspberry Pi. El sistema implementado permite la integración de sensores ESP32 para el monitoreo ambiental y un botón de emergencia para generar alertas en tiempo real a través de servicios como Telegram. Los datos se procesan mediante FastAPI y se almacenan en InfluxDB, garantizando autonomía y seguridad en la gestión de la información. Las pruebas en un entorno simulado demostraron una respuesta rápida, bajo consumo de recursos y recuperación automática ante fallos. La solución presentada es escalable, flexible y adaptable, ideal para aplicaciones en hogares o centros de asistencia, con posibilidades de expansión futura hacia inteligencia artificial y entornos reales.

Palabras clave: Internet de las Cosas (IoT); microservicios; Docker; Raspberry Pi; monitoreo local; arquitectura descentralizada

Abstract

The automation of smart environments improves safety and quality of life for vulnerable individuals, such as the elderly. However, many existing solutions rely heavily on cloud services, increasing costs and compromising privacy and latency. This work proposes a decentralized architecture based on microservices and Docker containers, designed to operate locally using low-cost hardware such as the Raspberry Pi. The implemented system allows the integration of ESP32 sensors for environmental

monitoring and an emergency button to trigger real-time alerts through services like Telegram. Data is processed via FastAPI and stored in InfluxDB, ensuring autonomy and secure data management. Tests conducted in a simulated environment demonstrated fast response times, low resource consumption, and automatic recovery from failures. The proposed solution is scalable, flexible, and adaptable—ideal for applications in homes or care centers—with future expansion potential towards artificial intelligence and deployment in real-world scenarios.

Keywords: Internet of Things (IoT); microservices; Docker; Raspberry Pi; local monitoring; decentralized architecture

1. Introducción

El avance en tecnologías de automatización ha permitido el desarrollo de entornos inteligentes que mejoran la seguridad y calidad de vida de las personas, especialmente aquellas en situaciones de vulnerabilidad, como los adultos mayores. En este contexto, la implementación de sistemas de monitoreo y asistencia basados en el Internet de las Cosas (IoT) ha crecido significativamente, facilitando la supervisión remota y la respuesta ante emergencias (Alam, Reaz, & Ali, 2012)

Sin embargo, muchas de estas soluciones dependen de servicios en la nube, lo que introduce diversos desafíos, como costos elevados, riesgos en la privacidad de los datos y latencia en la respuesta (Gubbi et al., 2013). El almacenamiento y procesamiento de datos en servidores remotos puede comprometer la seguridad de la información personal y generar tiempos de respuesta inadecuados en situaciones críticas (Da Xu, He, & Li, 2014). Además, la dependencia de una conexión a internet estable puede limitar la funcionalidad del sistema en entornos con conectividad deficiente (Al-Fuqaha et al., 2015).

Para abordar estas limitaciones, en este trabajo se propone una arquitectura descentralizada basada en microservicios y Docker, diseñada para operar en casi su totalidad de manera local, pero con la flexibilidad de que, al agregar otro nodo, puede extenderse hacia la nube. Esta arquitectura minimiza los costos operativos y elimina la dependencia de servicios en la nube, garantizando la privacidad y la disponibilidad de los datos en todo momento. Adicionalmente, la arquitectura permite la integración con sistemas de mensajería para el envío de alertas en tiempo real, mejorando la capacidad de respuesta ante eventos críticos.

El sistema propuesto se implementa en un entorno de pruebas que, si bien no involucra pacientes reales, replica físicamente las condiciones y distribución de un espacio monoambiente real. En este entorno se emplean sensores ESP32 para la monitorización de variables ambientales, así como un botón de emergencia para notificaciones inmediatas. Los datos son procesados por un servidor local en una Raspberry Pi, donde se gestionan con FastAPI y se almacenan en InfluxDB. Cada componente opera como un microservicio independiente, lo que facilita la escalabilidad y flexibilidad del sistema. Además, la integración con Docker permite la incorporación de nuevos servicios o actuadores sin afectar la estabilidad del sistema.

Este artículo está estructurado de la siguiente manera: en la Sección 2 se presenta el estado del arte, donde se revisan soluciones previas y su comparación con la propuesta; en la Sección 3 se describe la metodología empleada en el diseño e implementación del sistema; en la Sección 4 se presentan los resultados de la evaluación del sistema; y finalmente, en la Sección 5 se discuten las conclusiones y posibles líneas de trabajo futuro.

2. Estado del Arte

La automatización de espacios inteligentes ha sido ampliamente investigada en la última década, especialmente en el contexto de la asistencia a personas mayores. Los sistemas tradicionales basados en la nube, como Google Home, Amazon Alexa o Apple HomeKit, ofrecen potentes capacidades de procesamiento, pero presentan limitaciones relacionadas con la privacidad de los datos y la dependencia de la conectividad externa (Zeng et al., 2017).

En respuesta a estas limitaciones, han surgido propuestas que promueven arquitecturas locales o híbridas. Por ejemplo, Alam et al. (2012) destacan la importancia de sistemas inteligentes que funcionen localmente para garantizar baja latencia y autonomía en la toma de decisiones (Alam et al., 2012). Estas arquitecturas buscan reducir la dependencia de plataformas comerciales centralizadas y permitir un mayor control sobre los datos.

En paralelo, el uso de microservicios ha ganado relevancia en el desarrollo de aplicaciones IoT debido a su capacidad de modularizar funcionalidades y facilitar la escalabilidad de los sistemas (Villamizar et al., 2015). En el contexto de hogares inteligentes, los microservicios permiten que sensores, actuadores y servicios de procesamiento funcionen de forma independiente, lo cual resulta esencial para mantener la robustez del sistema ante posibles fallas.

Asimismo, la incorporación de contenedores Docker en entornos de bajo consumo como Raspberry Pi ha permitido ejecutar múltiples servicios de manera aislada, ligera y eficiente, siendo una alternativa viable para el despliegue local de soluciones IoT (Merkel, 2014). Docker facilita la portabilidad del sistema, lo que permite su rápida replicabilidad en otros entornos sin necesidad de reconfiguración compleja.

Finalmente, estudios como el de Dinh et al. (2022) han demostrado el uso exitoso de Telegram como canal de notificación en sistemas inteligentes, debido a su API flexible, disponibilidad multiplataforma y facilidad de integración con scripts y servicios backend (Dinh et al., 2022).

Estas soluciones y tecnologías constituyen la base sobre la cual se diseña la arquitectura propuesta en este trabajo, la cual busca combinar los beneficios de los microservicios, el despliegue local y la modularidad con una infraestructura de bajo costo y alta disponibilidad.

3. Metodología

La metodología desarrollada para este proyecto sigue un enfoque modular y descentralizado, fundamentado en los principios de la arquitectura de microservicios. Cada componente del sistema



fue diseñado, implementado y desplegado como un servicio independiente utilizando contenedores Docker, con el objetivo de facilitar la escalabilidad, la mantenibilidad y la portabilidad del sistema.

3.1. Arquitectura general

La figura 3.1 presenta la arquitectura IoT distribuida en cinco capas funcionales: adquisición, comunicación, procesamiento, almacenamiento y servicio. Cada capa cumple un rol específico, desde la recolección de datos mediante sensores conectados a microcontroladores ESP32, hasta su transmisión mediante MQTT, procesamiento en un servidor local desarrollado con FastAPI, almacenamiento en InfluxDB y generación de alertas o visualización a través de servicios como Telegram.

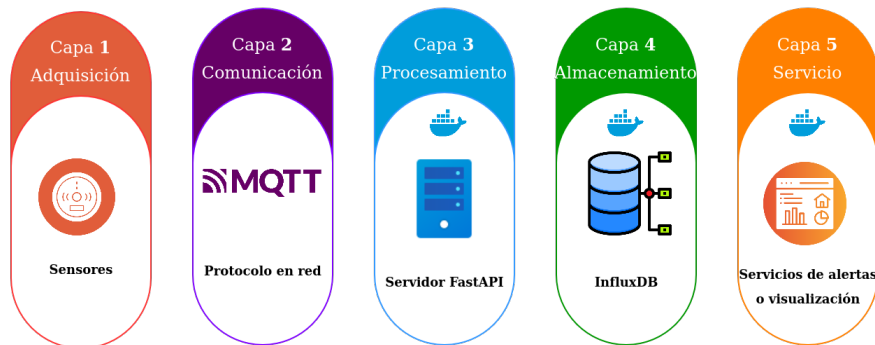


Figura 3.1 Capas de la arquitectura (Elaboración propia)

La figura 3.2 detalla el flujo de información dentro del sistema. Se observa cómo los datos capturados por sensores mediante MQTT a través de una red interna. Posteriormente, el servidor local —contenedorizado con Docker— gestiona el procesamiento y almacenamiento de datos, permitiendo su consulta y diagnóstico. Finalmente, el sistema puede notificar al usuario mediante Telegram, gracias a un agente automatizado de alertas.

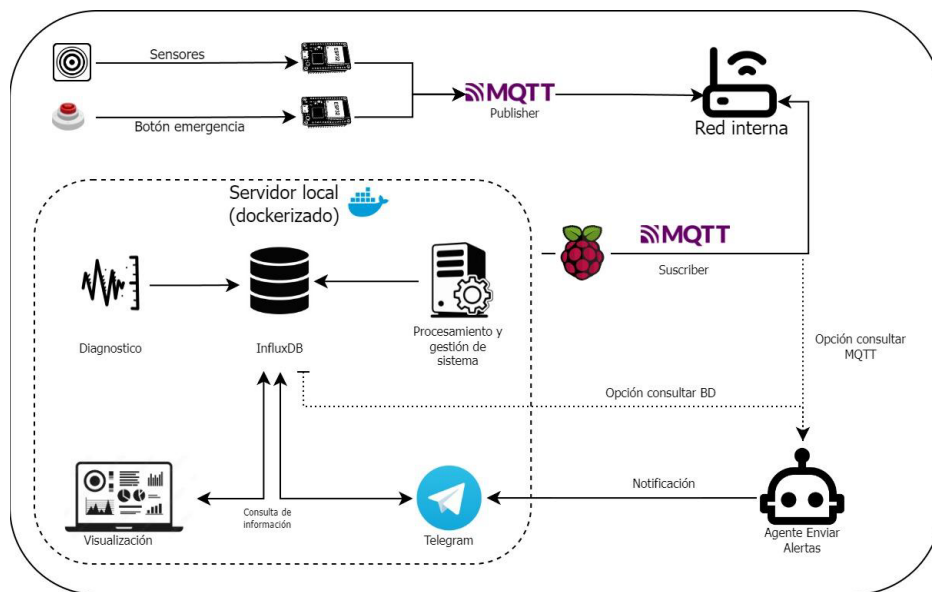


Figura 3.2 Diagrama de la arquitectura (Elaboración propia)

3.2. Comunicación y procesamiento de datos

Los sensores ESP32 están programados para enviar los datos a un broker MQTT local instalado en la Raspberry Pi. Un microservicio desarrollado con FastAPI se suscribe a los tópicos relevantes (ver tabla 3.3) y procesa los datos entrantes. Estos datos se almacenan en InfluxDB, permitiendo consultas por rango de tiempo y análisis históricos.

Tópico MQTT	Frecuencia de envío	Tipo de dato	Dispositivo de origen
temperatura	4s	Númerico	Esp32_Temp
distancia	1s	Númerico	Esp32_Dist
boton	Evento	Binario (1)	Esp32_btn

Tabla 3.3 Descripción de la frecuencia de transmisión de datos (Elaboración propia)

El botón de emergencia envía una señal específica al broker, que es capturada por otro microservicio encargado de emitir alertas. Este microservicio verifica el tipo de evento y envía mensajes al canal de Telegram configurado. También puede configurarse para otros servicios como correo electrónico o SMS.

3.3. Contenerización y despliegue

Cada microservicio está encapsulado en un contenedor Docker con su propia configuración y dependencias, lo que garantiza su funcionamiento de forma autónoma. Para su despliegue, se clona el repositorio correspondiente y se personaliza un archivo de configuración en formato JSON, el cual contiene los parámetros necesarios para el funcionamiento del servicio (como claves de autenticación, tópicos MQTT, etc). Posteriormente, se ejecuta el Dockerfile asociado para construir la imagen del contenedor y hacer la ejecución (ver Figura 3.4).

Como opción, el sistema completo básico puede ser orquestado mediante Docker Compose, lo que permite automatizar el despliegue de todos los microservicios en la Raspberry Pi u otros dispositivos compatibles. Esta estrategia garantiza la independencia entre servicios, permitiendo reinicios, actualizaciones o reemplazos sin afectar al resto del sistema. Además, los contenedores son portables y replicables, lo que favorece su adopción en diferentes entornos, especialmente en infraestructuras de bajo consumo energético como las que se implementan en entornos IoT (Merkel, 2014).

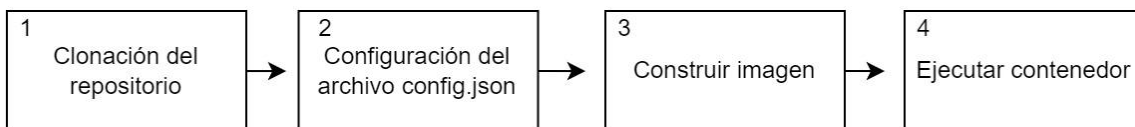


Figura 3.4 Diagrama de flujo del despliegue de los servicios (Elaboración propia)

3.4. Seguridad y privacidad

Al operar en una red local sin conexión constante a la nube, el sistema reduce significativamente el riesgo de exposición de datos sensibles. En esta etapa, se han implementado medidas de seguridad básicas, como reglas de firewall a nivel del sistema operativo y autenticación simple para el acceso a la API. Estas medidas permiten controlar el acceso y proteger los servicios más críticos.

Aunque el sistema aún no incluye cifrado avanzado ni mecanismos de autenticación robustos, la arquitectura está preparada para incorporar, en fases futuras, protocolos como TLS y autenticación basada en tokens (por ejemplo, JWT), con el fin de reforzar la seguridad sin comprometer la eficiencia en entornos de bajo consumo. La Tabla 3.5 resume las medidas de seguridad actualmente implementadas, sus propósitos y posibles mejoras que se pueden considerar en futuras versiones del sistema.

Medida implementada	Propósito	Mejora futura sugerida	Objetivo adicional
Reglas de firewall	Limitar el acceso a puertos y servicios no necesarios	Configuración avanzada de firewall (iptables, ufw)	Aislar servicios por rango IP o interfaz
Autenticación básica en la API	Restringir el acceso no autorizado	Autenticación basada en tokens (JWT, OAuth2)	Mayor control y trazabilidad de usuarios
Red local aislada	Minimizar exposición a internet	VPN para acceso remoto seguro	Acceso cifrado a la infraestructura desde fuera
Sincronización manual de servicios	Evitar conexiones externas innecesarias	Auditoría de logs y monitoreo de eventos	Detectar comportamientos anómalos o intentos de intrusión

Tabla 3.5 Medidas de seguridad implementadas y mejoras sugeridas

Esta metodología asegura un diseño robusto, autónomo, escalable y adaptable, con la posibilidad de extenderse para incluir nuevos sensores, actuadores y funcionalidades según las necesidades del entorno inteligente.

4. Resultados y Evaluación

Para validar la efectividad del sistema propuesto, se realizaron pruebas en un entorno controlado que simula un espacio habitado por una persona mayor. Las pruebas se centraron en evaluar la latencia de respuesta, el consumo de recursos en la Raspberry Pi, la estabilidad de los microservicios y la capacidad del sistema para operar de forma autónoma sin conexión a internet.

4.1. Escenario de prueba

El sistema fue desplegado en una Raspberry Pi 4 Model B con 8 GB de RAM, equipada con un procesador Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit a 1.5 GHz, ejecutando todos los contenedores Docker necesarios. Se utilizaron tres sensores ESP32 para adquirir lecturas de temperatura, vibración y distancia, así como un botón físico conectado a otro ESP32 para generar eventos de emergencia (ver Figura 4.1).

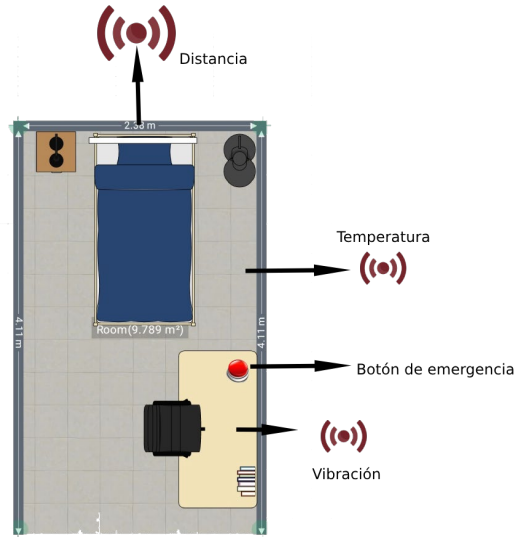


Figura 4.1 Escenario de prueba (Elaboración propia)

4.2. Tiempo de Respuesta del Sistema

El tiempo medio de respuesta desde que un evento (como la activación del botón) es generado hasta que la alerta llega a Telegram fue de aproximadamente 1.6 segundos. El tiempo de integración de nuevos sensores (desde la conexión física hasta su reconocimiento e inicio de transmisión de datos) fue de 4 segundos en promedio. Además, el sistema mostró una capacidad de recuperación completa ante fallos en menos de 8 segundos, incluyendo la reconexión automática al broker MQTT y la reactivación del microservicio correspondiente. La Tabla 4.2 muestra los tiempos y resultados más relevantes obtenidos durante las pruebas clave de funcionamiento del sistema.

Tipo de evento	Tiempo promedio de respuesta	Prueba realizada	Condiciones	Resultados
Botón de emergencia.	1.6s	Activación del botón de emergencia.	Botón pulsado manualmente.	Alerta recibida en 1.6s.
Integración de nuevos sensores.	4 segundos	Integrar sensor de vibración.	Tener configurado el tópicos mqtt.	Sensor identificado, datos recibidos y almacenados correctamente.
Recuperación completa ante fallos del sistema.	8s	Reinicio manual de contenedor Docker.	Contenedor detenido y reiniciado durante operación.	Servicio reiniciado, comunicación y procesamiento reanudados.

Tabla 4.2 Resultados de pruebas de desempeño del sistema

4.3. Consumo de recursos

Durante el monitoreo continuo por un período de 24 horas, el consumo de CPU promedio se mantuvo por debajo del 35% y el uso de RAM no superó el 10%. Estos valores validan la viabilidad del sistema en hardware de bajo consumo.

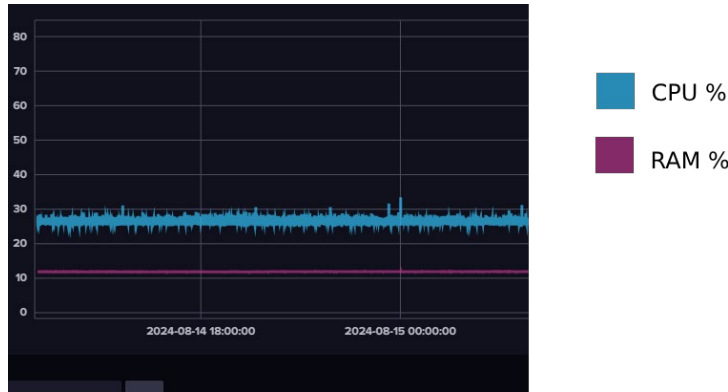


Tabla 4.3 Consumo de recursos durante 24 horas (Elaboración propia)

4.4. Estabilidad y Recuperación ante Fallos

Se simularon reinicios aleatorios de contenedores y pérdida temporal de conexión entre sensores y servidor. En todos los casos, los microservicios se recuperaron de manera autónoma y se restableció la operación sin intervención manual, lo cual refuerza la robustez de la solución basada en Docker. Cabe destacar que los contenedores Docker de cada microservicio fueron configurados con la opción *restart: always*, lo que permite que los servicios se reinicien automáticamente en caso de fallos o reinicios del sistema. Esta característica fue clave para garantizar la recuperación autónoma observada durante las pruebas (ver Tabla 4.4).

Evento simulado	Tiempo de recuperación	Mecanismo de recuperación
Reinicio de microservicios	< 5 segundos	Docker `restart: always`
Desconexión de sensores ESP32	~10 segundos	Reintento automático vía MQTT

Tabla 4.4 Pruebas de recuperación ante fallos (Elaboración propia)

5. Conclusiones y Trabajo Futuro

La arquitectura propuesta demuestra ser una solución eficaz, robusta y económica para la automatización de entornos inteligentes, especialmente orientada al cuidado de personas mayores. Al operar en una red local mediante una estructura basada en microservicios y contenedores Docker, el sistema garantiza baja latencia, alta disponibilidad y un mayor control sobre la privacidad de los datos.

Las pruebas realizadas confirmaron tiempos de respuesta adecuados para escenarios críticos, como la activación del botón de emergencia, así como una integración sencilla y rápida de nuevos sensores. El sistema mostró una notable estabilidad, incluso ante fallos intencionados, y demostró su capacidad de operar de manera autónoma sin conexión permanente a internet.

El uso de hardware de bajo costo como la Raspberry Pi, combinado con tecnologías modernas como FastAPI, InfluxDB, y servicios de mensajería como Telegram, permite replicar esta arquitectura en distintos contextos sin requerimientos técnicos o económicos elevados.

Como parte de la evolución del sistema, se espera en trabajo futuro desarrollar lo siguiente:

- Incorporar análisis predictivo con modelos de inteligencia artificial para detectar patrones de comportamiento anómalos.
- Expandir la red de sensores y actuadores, incluyendo dispositivos como cerraduras inteligentes o sistemas de control de luz.
- Implementar una interfaz gráfica multiplataforma que facilite la configuración del sistema y la visualización de datos por parte de usuarios y cuidadores.
- Evaluar el sistema en un entorno real con usuarios finales, con el fin de validar su impacto en la calidad de vida de los usuarios y la eficiencia del monitoreo.

6. Agradecimientos

Los autores quieren agradecer el apoyo del Tecnológico Nacional de México Campus León, así como al Consejo Nacional de Humanidades, Ciencias y Tecnologías para el desarrollo de este proyecto.

7. Referencias

Artículos de revistas

- Alam, M. R., Reaz, M. B. I., & Ali, M. A. M. (2012). A review of smart homes—Past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), 1190–1203. <https://doi.org/10.1109/TSMCC.2012.2189204>
- Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. *Linux Journal*, (239), 2. Retrieved from <https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment>
- Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks*, 76, 146–164. <https://doi.org/10.1016/j.comnet.2014.11.008>.

Memorias del congreso

- Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R. and Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *Proceedings of the 10th Computing Colombian Conference (10CCC)*, Bogotá, pp. 583–590.

- Yousuf, M., Asad, M. and Ameen, M. (2020). Smart Alert System using Telegram for Home Automation. *Proceedings of the IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pp. 1–4.

Fuentes electrónicas

- Zeng, E., Mare, S. and Roesner, F. (2017). End user security and privacy concerns with smart homes. *Proceedings of the Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pp. 65–80. Consultado el 21 de abril de 2025 en <https://www.usenix.org/conference/soups2017/technical-sessions/presentation/zeng>

Sobre los autores

- **César Abraham Delgado Cardona**, Ingeniero en Sistemas Computacionales y estudiante de tiempo completo de la Maestría en Ciencias de la Computación en el Tecnológico Nacional de México Campus León, México. Email: cesardelgadocardona@gmail.com
- **Carlos Lino Ramírez**, Doctor en Computación. Jefe de Centro de cómputo en Instituto Tecnológico de León, México. Miembro del Cuerpo Académico de Bioinformática y Tecnologías Computacionales. Email: carlos.lino@leon.tecnm.mx
- **Víctor Manuel Zamudio Rodríguez**, Doctor en Ciencias de la Computación. Profesor-investigador titular del Instituto Tecnológico de León, México. Miembro del Sistema Nacional de Investigadoras e Investigadores SNII Nivel 1, y miembro del Cuerpo Académico de Bioinformática y Tecnologías Computacionales. Email: vic.zamudio@leon.tecnm.mx
- **David Asael Gutiérrez Hernández**, Doctor en Física. Profesor-investigador titular del Instituto Tecnológico de León, México. Miembro del Sistema Nacional de Investigadoras e Investigadores de México, SNII Nivel 2. Miembro del cuerpo académico Bioinformática y Tecnologías Computacionales. Email: david.gutierrez@leon.tecnm.mx
- **Rafael Santos Pérez**, Maestro en Ciencias de la Computación por el Instituto Tecnológico de León, candidato a Doctor en Ingeniería en Tecnologías Emergentes por la Universidad Cristóbal Colón, México. Actualmente desempeña la posición de Professionnel de Recherche en la Université de Moncton, Canadá. Email: rafael.santos.perez@umoncton.ca

Los puntos de vista expresados en este artículo no reflejan necesariamente la opinión de la Asociación Colombiana de Facultades de Ingeniería.

Copyright © 2025 Asociación Colombiana de Facultades de Ingeniería (ACOFI)