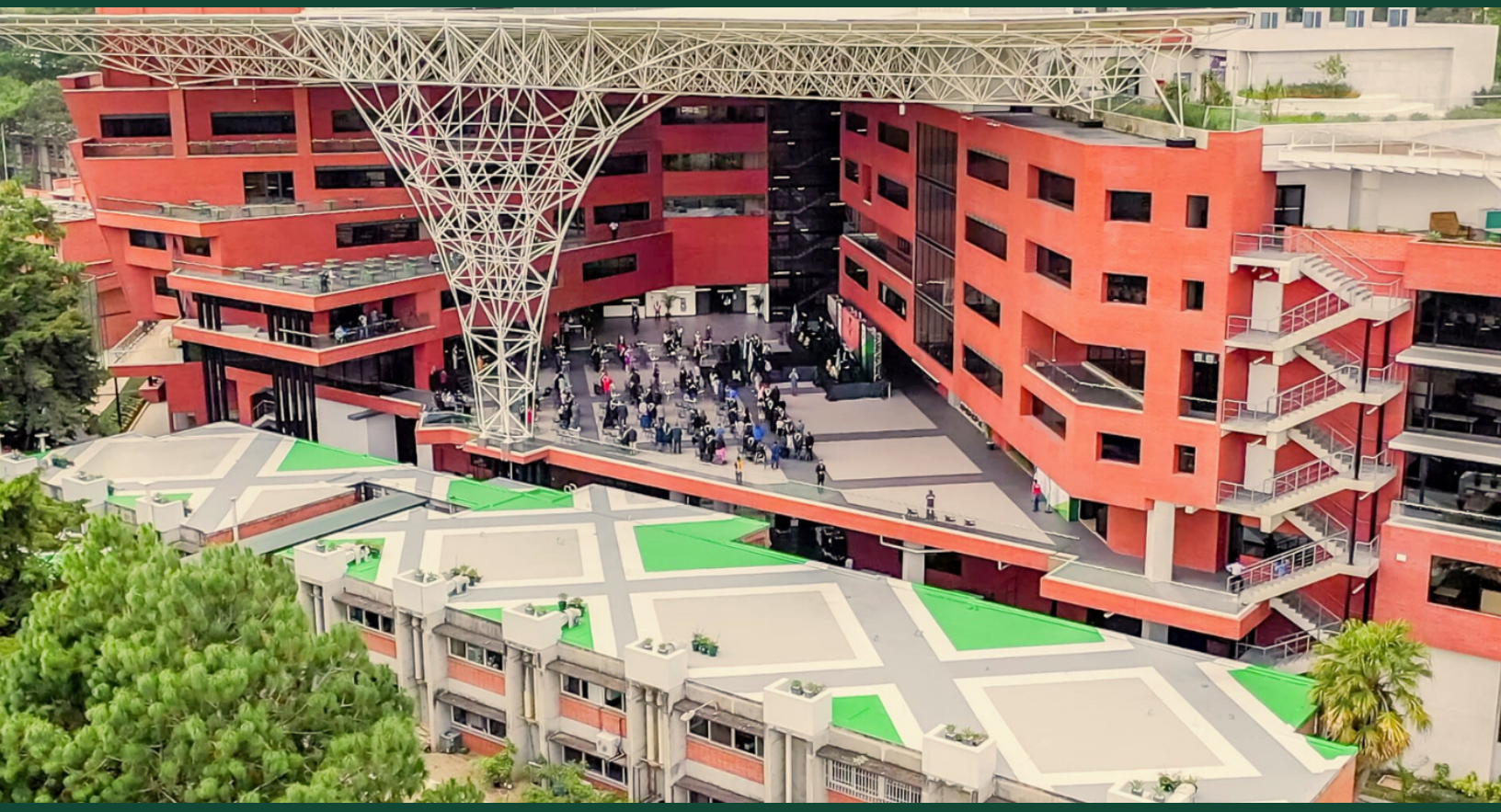
Diseño e implementación de infraestructura de software para la conexión remota sincrónica con el laboratorio Robotat de la Universidad del Valle de Guatemala

Sara Ximena Hernández Recinos



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Diseño e implementación de infraestructura de software para la conexión remota sincrónica con el laboratorio Robotat de la Universidad del Valle de Guatemala

Trabajo de graduación presentado por Sara Ximena Hernández Recinos para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

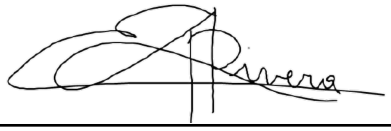
Diseño e implementación de infraestructura de software para la conexión remota sincrónica con el laboratorio Robotat de la Universidad del Valle de Guatemala

Trabajo de graduación presentado por Sara Ximena Hernández Recinos para optar al grado académico de Licenciada en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:



(f) _____

Dr. Luis Alberto Rivera Estrada

(f) _____

M.Sc. Carlos Esquit Hernández

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

A lo largo de este proyecto, he tenido la oportunidad de crecer no solo en el ámbito académico, sino también en lo personal. Culminar este trabajo representa para mí la suma de esfuerzo, disciplina y, sobre todo, del apoyo incondicional de personas fundamentales en mi vida.

En primer lugar, quiero agradecer a Dios, fuente de fortaleza y esperanza, por darme la claridad y la perseverancia necesarias para no rendirme en los momentos más desafiantes. Su presencia constante ha sido un sostén en este camino, recordándome que todo esfuerzo tiene un propósito mayor. Junto a Él, guardo en mi corazón a mi abuelita Oly cuyo cariño y fe me acompañaron desde niña. A ella le hubiera gustado verme llegar hasta aquí; por eso, hoy la llevo conmigo en silencio y con afecto.

A mis padres, Vannessa y José Miguel, pilares esenciales de mi vida, les debo una gratitud eterna. Su amor incondicional, sus consejos y la confianza que siempre han depositado en mí me impulsaron a alcanzar cada meta que me propuse. A mi hermana Paula, gracias por su apoyo inquebrantable y por poner sus manos y su corazón cuando mis ideas necesitaban forma. No puedo dejar de mencionar a Bailey, mi fiel compañera de cuatro patas, cuya lealtad y alegría fueron un bálsamo emocional durante las etapas más intensas de este trabajo.

En el ámbito académico, también encontré figuras clave. Al Dr. Luis Rivera, mi asesor, le agradezco por compartir su conocimiento y guiarme con paciencia, profesionalismo y compromiso durante el desarrollo de este trabajo. Sus observaciones y sugerencias fueron clave para dar solidez a este proyecto. Asimismo, agradezco a MSc. Miguel Zea por su constante acompañamiento, motivación y respaldo, elementos determinantes en cada etapa de este proceso.

Este trabajo no habría sido posible sin cada uno de ellos, quienes, con sus distintos aportes, me ayudaron a transformar una idea en un proyecto tangible. A todos, gracias por ser parte de este logro, que marca una etapa trascendental en mi vida académica y personal.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El presente trabajo desarrolla una infraestructura de *software* destinada a habilitar la conexión remota sincrónica con el laboratorio Robotat de la Universidad del Valle de Guatemala. El proyecto surge de la necesidad de superar las limitaciones asociadas al acceso exclusivamente presencial, las cuales restringen la disponibilidad del laboratorio y dificultan la continuidad de prácticas experimentales. Para atender esta problemática, se planteó como objetivo principal diseñar e implementar una arquitectura local capaz de integrar dispositivos heterogéneos: robots móviles, brazos manipuladores, cámaras IP y sistema de captura de movimiento, mediante servicios centralizados y mecanismos de comunicación estandarizados.

La metodología se dividió en tres componentes. En primer lugar, se reestructuró la infraestructura local mediante la incorporación de un servidor dedicado para la gestión de dispositivos, servicios web y mensajería. En segundo lugar, se implementó un módulo de monitoreo y transmisión de video basado en cámaras Amcrest, integrando control PTZ y flujos `MJPEG`. Finalmente, se diseñó un sistema de autenticación orientado a la administración de usuarios institucionales, sentando las bases para su futura extensión a esquemas remotos mediante VPN y proxy inverso.

Los resultados obtenidos evidencian que la consolidación local permite gestionar dispositivos, transmitir video en tiempo real y centralizar servicios sin comprometer la latencia. Además, se establecieron lineamientos técnicos para una arquitectura remota futura con acceso seguro, la cual podrá implementarse de forma incremental sobre la base construida. El proyecto demuestra que una infraestructura local bien diseñada constituye el punto de partida necesario para habilitar un laboratorio de robótica accesible a distancia.

Palabras clave: Robotat, protocolos de red, conexión remota, sincrónica, latencia.

Abstract

A software infrastructure was developed to enable synchronous remote connection with the Robotat laboratory at Universidad del Valle de Guatemala. The project arises from the need to overcome the limitations of exclusively in-person access, which restrict laboratory availability and hinder the continuity of experimental practices. To address this issue, the main objective was to design and implement a local architecture capable of integrating heterogeneous devices such as mobile robots, robotic arms, IP cameras, and the motion-capture system through centralized services and standardized communication mechanisms.

The methodology was divided into three components. First, the local infrastructure was restructured through the incorporation of a dedicated server for device management, web services, and messaging. Second, a monitoring and video-streaming module based on Amcrest cameras was implemented, integrating PTZ control and `MJPEG` streams. Finally, an authentication system oriented toward the management of institutional users was designed, laying the groundwork for future extension to remote access schemes through VPN and reverse proxy.

The results show that the local consolidation enables device management, real-time video transmission, and centralized services without compromising latency. Furthermore, technical guidelines were established for a future remote architecture with secure access, which may be implemented incrementally on top of the foundation established in this work. The project demonstrates that a well-designed local infrastructure constitutes the necessary starting point for enabling a remotely accessible robotics laboratory.

Keywords: Robotat, network protocols, remote connection, synchronous, latency.

CAPÍTULO 1

Introducción

El desarrollo de laboratorios remotos en el ámbito de la ingeniería ha cobrado una relevancia creciente, impulsado por la necesidad de ampliar el acceso a infraestructura tecnológica especializada más allá de las limitaciones físicas y geográficas. En este contexto, el presente trabajo se orienta al diseño e implementación de una infraestructura de software que permita la conexión remota sincrónica con el laboratorio Robotat de la Universidad del Valle de Guatemala. El problema central que motiva este proyecto radica en la ausencia de una arquitectura integrada que, además de posibilitar el control remoto de robots y sistemas de captura de movimiento, garantice seguridad en el acceso, eficiencia en la transmisión de datos y disponibilidad de video en tiempo real. Estas carencias limitan la versatilidad, escalabilidad y confiabilidad del laboratorio como recurso educativo y de investigación.

Con el objetivo de superar estos desafíos, se propone una solución integral que siente las bases de un laboratorio remoto robusto, seguro y funcional. Para ello, se definieron los siguientes objetivos específicos: (1) reestructurar la infraestructura de red local; (2) implementar transmisión de video en tiempo real con control de cámaras PTZ; (3) diseñar un sistema de autenticación basado en roles y tokens JWT con mecanismos de revocación segura; y (4) evaluar alternativas de configuración de red que sustenten una arquitectura remota escalable. Cada uno de estos componentes responde a la necesidad de transformar al Robotat en un entorno digital más accesible, eficiente y protegido.

El alcance del trabajo se centra exclusivamente en la infraestructura de software y comunicación, dejando fuera aspectos relacionados con el diseño mecánico, hardware específico o algoritmos avanzados de control robótico, los cuales se consideran ya disponibles en el laboratorio. La metodología empleada incluyó un diagnóstico inicial de la red existente, la integración de servicios en un servidor local centralizado, el desarrollo de una plataforma web con gestión de roles y transmisión de video, y la aplicación de técnicas de análisis multicriterio para comparar arquitecturas de

conexión remota.

Los resultados obtenidos demuestran que la consolidación de servicios mediante un *broker* MQTT (`Mosquitto`) permite una comunicación eficiente, organizada y escalable entre dispositivos, mientras que la estandarización de los mensajes asegura consistencia y trazabilidad. Asimismo, la implementación de flujos RTSP optimizados (con `subtype=1`) reduce significativamente el consumo de ancho de banda y mejora la latencia, lo que resulta crítico para la experiencia remota. Por otro lado, el sistema de autenticación basado en tokens JWT y una lista negra de revocación garantiza un control de acceso seguro y dinámico, capaz de responder ante eventos como cierre de sesión o cambio de credenciales. Finalmente, la arquitectura *frontend-backend* (`React/Vite y Django`) no solo ofrece una interfaz visual coherente y de alto rendimiento, sino que también centraliza la lógica de autorización por roles, reforzando la seguridad del sistema.

Este documento se estructura en capítulos alineados con los objetivos específicos: tras los apartados introductorios (antecedentes, justificación, planteamiento del problema y marco teórico), se presentan los resultados de la implementación local, la descripción del sistema de autenticación y la evaluación comparativa de arquitecturas de red, culminando con un conjunto de conclusiones y recomendaciones para futuras extensiones del laboratorio remoto.

CAPÍTULO 2

Antecedentes

En la última década, los laboratorios remotos han adquirido relevancia significativa dentro del ámbito educativo, especialmente en carreras de ingeniería. En la pandemia del COVID-19, surgió la necesidad de brindar experiencias prácticas accesibles, asincrónicas y escalables. Los laboratorios remotos han permitido que las personas interactúen con *hardware* físico real desde ubicaciones geográficas distintas, utilizando plataformas digitales. Esta modalidad no solo responde a situaciones de emergencia sanitaria, sino que también abre nuevas posibilidades para la educación a distancia, el uso compartido de recursos y la expansión del acceso a la formación práctica en ingeniería [1]. A lo largo de los años, se han desarrollado plataformas internacionales tales como el Robotarium del Instituto de Tecnología de Georgia [2], WebLab-Deusto desarrollado por la Universidad de Deusto [3] y RExLab creado por la Universidad Federal de Santa Catarina en Brasil [4]. Estos entornos digitales han demostrado que es factible y beneficioso ofrecer acceso remoto a laboratorios físicos para fines educativos y de investigación.

El Robotarium (Figura 1) es una plataforma de investigación en robótica de enjambres que permite a los usuarios de todo el mundo ejecutar algoritmos en robots físicos de manera remota. Su arquitectura está compuesta por varios subsistemas clave. La infraestructura física cuenta con robots y un área de operación. En el área de operación se delimita un espacio físico donde los robots realizan tareas programadas, equipado con sistemas de seguimiento y monitoreo. El siguiente subsistema en la arquitectura de la plataforma es el servidor central. Este actúa como intermediario entre los usuarios remotos y los robots, gestionando a la recepción de códigos, su validación y la asignación de tareas a los robots. La red de comunicación implementa protocolos de comunicación inalámbrica para la transmisión de datos entre el servidor y los robots. En cuanto a la interfaz del usuario, el portal web permite a los usuarios cargar sus algoritmos, programar experimentos y visualizar resultados [2].

Figura 1. Vista general del laboratorio Robotarium (Georgia Tech).

Nota. Plataforma de investigación que facilita la programación y control remoto de múltiples robots. Imagen obtenida de [5].

Desde su creación en 2017, el Robotarium se ha mantenido en evolución constante con el propósito de mejorar su funcionalidad y accesibilidad. Originalmente, el sistema utilizaba cámaras USB y marcadores para el seguimiento visual. En 2018, se actualizó con un sistema de captura de movimiento, que aumentó significativamente la precisión del rastreo, especialmente al trabajar con enjambres. En 2019, se integraron contenedores Docker para ejecutar los algoritmos enviados por los usuarios en entornos aislados, estandarizados y reproducibles. Además, se desarrolló un sistema de verificación automática del código como medida de seguridad para evitar colisiones y daños al *hardware*. En cuanto al manejo de los robots, no se comunican directamente entre ellos, sino que cada uno envía y recibe datos al/desde un controlador central que gestiona toda la lógica de control. Con respecto al hardware, los robots GRITSBot X fueron rediseñados en 2020 con mejoras como: carga inalámbrica inductiva, componentes modulares y capacidad de autogestión energética. En 2021, se incorporó un proyector de corto alcance para superponer información visual sobre el entorno físico y una cámara IP Ethernet para grabar los experimentos de manera automática. Asimismo, se rediseñó la plataforma web para permitir la programación, seguimiento y análisis de los experimentos, así como el acceso a simuladores y documentación [6].

Un segundo ejemplo de laboratorios remotos es el WebLab-Deusto desarrollado en la Universidad de Deusto. La arquitectura del *software* ha evolucionado significativamente. Al inicio, se usó un cliente desarrollado en C para la comunicación. Sin embargo, en versiones posteriores, se ha adoptado una arquitectura basada en tecnologías web 2.0, que mejoraron la interacción entre el cliente y el servidor. El cliente se ejecuta en cualquier navegador sin necesidad de *plug-ins* adicionales, lo que facilita el acceso desde diferentes plataformas. En cuanto al hardware, el laboratorio cuenta con una variedad de dispositivos para las prácticas de electrónica digital. Además, se han integrado otros dispositivos como microcontroladores [3].

WebLab-Deusto ofrece varios tipos de experimentos. Es posible realizar experimentos con robots móviles programables: se incorporan entornos visuales 3D como Second Life para programar los robots. También, se desempeñan laboratorios de electrónica e instrumentación, los cuales incluyen juegos de lógica y aplicaciones conectadas a

osciloscopios a través de GPIB. Por último, se pueden llevar a cabo laboratorios de WebLab-Submarine, un acuario real controlado de forma remota. Recientemente, se ha integrado realidad aumentada en el proyecto Boole-WebLab-Deusto, permitiendo superponer modelos virtuales sobre experimentos electrónicos con dispositivos hardware reales [7].

El Laboratorio de Experimentación Remota (RExLab) de la Universidad Federal de Santa Catarina es un referente en innovación educativa mediate tecnologías de acceso remoto. Fundado en 1997, combina hardware de bajo costo, plataformas de código abierto y comunicación basada en internet. El sistema de gestión de aprendizaje utiliza Moodle como entorno virtual para integrar experimentos remotos, materiales didácticos y seguimiento estudiantil. La plataforma de experimentación Relle, fue desarrollada internamente y gestiona el acceso a los experimentos físicos a través de interfaces web y móviles (RexMobile) [4]. Su diseño modular permite integración con sistemas de terceros. En cuanto al *hardware* experimental, cuenta con microservidores web los cuales son dispositivos embebidos de bajo costo que controlan sensores y actuadores en tiempo real.

RExLab cuenta con paneles eléctricos controlables remotamente y cámaras IP que transmiten video en vivo para observar los experimentos [8]. Adicionalmente, formaron alianzas con la Universidad Federal de Uberlandia y otras instituciones para crear una red nacional de laboratorios remotos interoperables. El modelo de este laboratorio prioriza soluciones de bajo costo y código abierto, facilitando su replicación en escuelas públicas [9].

En el caso de la Universidad del Valle de Guatemala (UVG), se cuenta con el sistema Robotat (Figura 2), un laboratorio físico especializado en robótica. A pesar de que su uso ha sido local, se proyecta una evolución hacia una infraestructura que incluya sistemas de seguimiento visual y control remoto. Como primer paso hacia esta transición, en la tesis de Jonathan Pu [10] se logró establecer un canal de comunicación funcional entre robots Pololu 3pi+ y una interfaz, permitiendo enviar comandos a los robots físicos, aunque aún requería el control presencial dentro del laboratorio. El proyecto sirvió como validación de conectividad local y representación visual (simulada) de los movimientos, sin embargo, no incluyó mecanismos de acceso remoto o asincrónico. Los laboratorios remotos antes mencionados, brindan modelos replicables que pueden inspirar la evolución del Robotat, con posibilidades de ser adaptados y mejorados según el contexto local.

Figura 2. Vista general del sistema Robotat en la Universidad del Valle de Guatemala.

Nota. El Robotat integra sistemas de captura de movimiento, robots móviles y brazos robóticos para experimentos. Imagen obtenida de [11].

CAPÍTULO 3

Justificación

El ecosistema Robotat en la Universidad del Valle de Guatemala ha impulsado el aprendizaje práctico en robótica a través del uso de agentes físicos como manipuladores, drones y robots móviles. No obstante, su uso ha estado restringido a entornos presenciales y horarios específicos dentro del campus central, lo cual limita el acceso a experiencias experimentales en diversas circunstancias. Existen múltiples situaciones en las que el acceso presencial no es posible, como lo evidenció la pandemia por COVID-19, donde la continuidad del aprendizaje práctico se vio comprometida por restricciones sanitarias. Asimismo, en contextos locales, eventos como bloqueos en carreteras, restricciones por seguridad o falta de transporte también afectan el acceso de los estudiantes a las instalaciones. Esta problemática se intensifica para estudiantes provenientes de departamentos alejados (Campus Sur UVG) o del extranjero, así como para aquellos con disponibilidad limitada debido a cargas laborales o académicas que les impiden asistir en horarios regulares.

Frente a estos desafíos, el uso de laboratorios remotos permite mantener la continuidad académica, democratizar el acceso a experiencias prácticas y promover la equidad en la educación. La plataforma Robotarium, permite la programación remota de robots físicos con herramientas de simulación previa, verificación de código y ejecución controlada. Implementar una metodología inspirada en la del Robotarium, con sus posibles áreas de mejora, dentro del contexto del Robotat no solo responde a una necesidad operativa, sino que representa un paso estratégico hacia una formación en robótica más flexible, inclusiva y resiliente. Además, fortalece competencias fundamentales como la autonomía, la autoorganización y pensamiento crítico, al permitir a los estudiantes gestionar sus propios experimentos y analizar resultados reales desde entornos controlados a distancia.

Este proyecto buscó extender las capacidades del Robotat al integrar un enfoque remoto, respondiendo de forma efectiva a los retos del acceso físico limitado y posicionado a la UVG como una institución adaptable a los desafíos contemporáneos de

la educación en ingeniería y robótica.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Desarrollar infraestructura de software que permita realizar una conexión sincrónica remota con el laboratorio Robotat.

4.2. Objetivos específicos

- Reestructurar la infraestructura de red local del Robotat, consolidando servicios existentes en un servidor dedicado.
- Implementar acceso al video en tiempo real de la infraestructura del Robotat, habilitar el control de cámaras físicas y definir cómo se obtendrá y validará la información del entorno.
- Diseñar una herramienta de autenticación que permita gestionar de forma segura el acceso a los usuarios del Robotat.
- Evaluar alternativas de configuración de red para fundamentar una propuesta de arquitectura orientada a una conexión sincrónica y remota eficiente.

CAPÍTULO 5

Alcance

El proyecto se enfocó en desarrollar la infraestructura de software necesaria para habilitar una futura conexión remota con el laboratorio Robotat de la Universidad del Valle de Guatemala. Esta delimitación se basó en recursos tecnológicos disponibles y las limitaciones de ejecución, priorizando acciones que permitirán establecer cimientos claros y funcionales para un sistema de acceso remoto en tiempo real. De esta manera, el trabajo se enfocó en construir una plataforma local funcional que pudiera evolucionar hacia un esquema de conexión más amplio.

El desarrollo se centró en la reestructuración de la red interna y la consolidación de servicios en un servidor dedicado. Asimismo, incluyó la habilitación de transmisión de video en tiempo real y el diseño de una herramienta de autenticación para la gestión segura de usuarios. Estos elementos fueron seleccionados por su impacto inmediato en el funcionamiento del laboratorio y por su potencial para integrarse en una arquitectura remota en fases posteriores. Aunque la conexión remota completa no se implementó, se propuso una solución técnica sustentada por la investigación realizada y pruebas previas.

Durante la ejecución se utilizaron pocos recursos físicos aparte de las cámaras Amcrest, ya que, el enfoque principal estuvo en el desarrollo de la infraestructura de software. El trabajo utilizó un servidor local para la centralizar los servicios, empleando herramientas como Python, Vite, React y Tailwind CSS, junto con bibliotecas especializadas como OpenCV y Paho MQTT. Se evitó el uso de infraestructura en la nube. Esto permitió trabajar en un entorno controlado y reproducible, además de validar configuraciones clave para el acceso remoto en etapas posteriores.

Los resultados incluyeron la integración funcional de los dispositivos bajo una infraestructura consolidada, la transmisión estable de video en tiempo real y un sistema de autenticación que gestionó de forma segura el acceso a los usuarios. Estas acciones no solo cumplieron con los objetivos planteados, sino que también establecieron las bases para una expansión futura hacia un sistema remoto sincrónico. En este sentido,

el alcance quedó definido como la construcción de una base operativa sólida que abre el camino a la siguiente fase del proyecto.

CAPÍTULO 6

Marco teórico

6.1. Protocolos de comunicación

Los protocolos de comunicación son conjuntos estructurados de reglas y convenciones que habilitan la interacción eficiente y coordinada entre sistemas diversos. Estos regulan aspectos como la sintaxis, semántica, control de errores y sincronización temporal [12]. En este contexto, los protocolos MQTT y TCP representan enfoques complementarios: mientras TCP establece una arquitectura sólida y confiable para el transporte de datos, MQTT aporta un mecanismo ligero y eficiente orientado a entornos con restricciones como el *IoT* [13], [14]. Su integración es vital para diseñar sistemas remotos sincrónicos que requieren tanto la integridad como la eficiencia en la transmisión de datos críticos.

6.1.1. TCP/IP

El protocolo TCP (Protocolo de Control de Transmisión) fue desarrollado en los años 70 como parte de un proyecto de investigación de ARPA con el propósito de interconectar redes de computadoras. ARPANET, la red experimental creada en este proyecto, demostró el éxito de la conmutación de paquetes y en 1976 ya operaba interconectando universidades y centros de investigación. La primera especificación consolidada fue publicada como RFC 793, definiendo TCP como un protocolo orientado a conexión para aplicaciones *multinetwork*, control de flujo y segmentación de datos. Desde entonces, este protocolo ha sido actualizado con extensiones como control de congestión y opciones de alto rendimiento. Estos avances consolidaron a TCP como el protocolo de transporte confiable más utilizado en Internet [15].

Gracias a esta evolución técnica, TCP opera actualmente en la capa de transporte y brinda un canal de comunicación confiable de extremo a extremo. La conexión es

establecida por medio de un *three-way handshake*, donde se intercambian los bits de control SYN y ACK para sincronizar números de secuencia y confirmar disponibilidad. Luego de estar activa la sesión, los datos son divididos en segmentos que incluyen cabeceras con campos de control como número de secuencia, número de acuse de recibo, *checksum* y tamaño de ventana [16].

La fiabilidad es asegurada a través de retransmisión de segmentos perdidos, confirmación mediante ACKs y uso de ventanas deslizantes para regular el flujo de datos entre emisor y receptor. Esta ventana permite ajustar dinámicamente cuántos bytes pueden enviarse sin recibir confirmación, evitando que un emisor rápido sature a un receptor lento. Además, TCP implementa control de congestión para adaptarse al estado de la red. Algoritmos tales como: *slow start*, *congestion avoidance* y *fast retransmit/fast recovery* ajustan la tasa de envío según las pérdidas detectadas y previenen el colapso de la red [17]. Un punto importante es que TCP provee un flujo de bytes continuo, no paquetes independientes. Esto significa que las aplicaciones no necesitan gestionar la fragmentación o el reordenamiento de datos: TCP se encarga de ensamblar el flujo en el orden correcto. En paralelo, maneja la detección de errores por medio de un *checksum* de 16 bits que cubre cabecera y datos, lo que permite descartar segmentos corruptos y solicitar su retransmisión. Esta arquitectura modular hace que TCP sea flexible y se adapte a diferentes tecnologías de red sin depender del hardware [18].

En redes locales, TCP actúa como el motor de los servicios que requieren comunicación confiable y persistente. Protocolos de aplicación como HTTP/HTTPS utilizan TCP para el acceso a páginas web, mientras que FTP y SMB/NFS lo hacen para la transferencia y compartición de archivos. En entornos empresariales, sistemas de gestión de bases de datos como MySQL o SQL Server dependen de TCP para mantener conexiones estables entre clientes y servidores [19].

Este protocolo desempeña un papel esencial en las conexiones remotas sincrónicas, donde la interacción entre cliente y servidor ocurre en tiempo real. Protocolos como Telnet y SSH funcionan sobre TCP para permitir que un usuario controle de forma remota un sistema como si estuviera físicamente presente. En estas sesiones, cada comando y respuesta viaja encapsulado en segmentos TCP, que garantizan la entrega ordenada, íntegra y sin duplicados [16].

En aplicaciones modernas, este mismo principio se extiende a herramientas de escritorio remoto (RDP o VNC) y sistemas de tele operación robótica, donde los datos de control deben llegar de forma fiable. En estos casos, TCP maneja la parte de control y señalización asegurando que los comandos enviados desde el usuario reciban correctamente por el sistema remoto. A pesar de que, protocolos como UDP son preferidos para transmisión de video o audio en tiempo real debido a su menor latencia, TCP sigue siendo indispensable para mantener la sesión interactiva estable y segura [18].

6.1.2. MQTT

MQTT (*Message Queuing Telemetry Transport*) es un protocolo de mensajería ligero basado en el modelo *publish/subscribe*, en el que los dispositivos publican o se suscriben a mensajes mediante un intermediario central conocido como *broker:*. Gracias a su simplicidad, eficiencia y adaptabilidad, MQTT se ha consolidado como uno de los protocolos predominantes en comunicaciones máquina a máquina ($M2M$) e Internet de las Cosas (IoT) [20].

En cuanto a su funcionamiento, opera sobre el mismo concepto del protocolo TCP/IP y se basa en tres componentes principales: los clientes, el *broker* o agente MQTT y los temas (*topics*) [21]. Los clientes pueden actuar como publicadores (*publishers*), enviando mensajes a temas específicos. Por otro lado, también pueden ejecutar el rol de suscriptor (*subscriber*), recibiendo mensajes de temas a los que se ha suscrito. Cabe destacar que, los clientes nunca se comunican directamente entre sí; toda la comunicación se realiza a través del *broker*, lo que permite una separación total de receptores y emisores. Los clientes comienzan la conexión con el *broker* enviando un mensaje *CONNECT* y mantienen la conexión activa mediante mensajes *PINGREQ/PINGRESP*, lo que facilita la administración eficiente de la red y permite que clientes y servidores funcionen de manera asíncrona y sin necesidad de estar conectados simultáneamente [22].

El *broker* es el núcleo de la arquitectura MQTT. Recibe todos los mensajes publicados y los redistribuye a los suscriptores interesados, gestionando además conexiones, autenticación, autorización, filtrado por temas y cuando se activa la persistencia de sesión el almacenamiento de mensajes pendientes. También puede incorporar funcionalidades avanzadas, como registro de mensajes para análisis posteriores o integración con sistemas *backend:* [23]. La topología típica es en estrella: todos los clientes se conectan al *broker* central, lo que facilita la escalabilidad y el control unificado de la comunicación [24].

Los *topics* o temas organizan y filtran la comunicación mediante cadenas de texto jerárquicas, similares a rutas de sistemas de archivos, por ejemplo, casa/piso1/salón/luz. Los clientes publican mensajes en un *topic* específico y otros se suscriben a él para recibir los mensajes correspondientes. Esta estructura admite comodines: el símbolo de suma representa un nivel único en la jerarquía y el numeral cubre múltiples niveles, permitiendo suscripciones flexibles a grupos amplios de mensajes. La jerarquía de temas refuerza el desacoplamiento entre publicadores y suscriptores, quienes no necesitan conocerse ni estar activos al mismo tiempo. Además, permite una segmentación granular de la información, ideal para entornos complejos como hogares inteligentes, ciudades conectadas o entornos industriales [25].

Este protocolo cuenta con tres niveles de calidad de servicio (QoS) que garantizan diferentes grados de fiabilidad en la entrega de mensajes. El nivel 0 es un envío sin confirmación (*"fire-and-forget"*), adecuado para datos no críticos. El nivel 1 garantiza que el mensaje se entregue por lo menos una vez por medio de confirmaciones simples, mientras que el nivel 2 asegura la entrega exactamente una vez a través de un proceso

de *handshake* en cuatro pasos; esto introduce mayor latencia y sobrecarga [26].

En cuanto a la seguridad, MQTT presenta limitaciones inherentes: carece de mecanismos de protección integrados, lo que expone a vulnerabilidades como la falta de validación de paquetes, omisión de campos obligatorios o errores lógicos en implementaciones que pueden derivar en ataques de denegación de servicio o accesos no autorizados. Para mitigarlos, se recomienda cifrado TLS/SSL, autenticación mutua y controles de acceso basados en roles. Actualmente, también se exploran soluciones avanzadas, como la integración de inteligencia artificial para detección de anomalías y textitblockchain para garantizar la integridad de los mensajes [26].

Existen diversos *brokers* MQTT, cada uno diseñado para adaptarse a diferentes entornos y requisitos. Entre los más utilizados se encuentran Eclipse Mosquitto, EMQX, HiveMQ, VerneMQ y NanoMQ. Aunque la mayoría de estos ofrecen funcionalidades avanzadas como *clustering*, soporte para múltiples protocolos y alta disponibilidad, Eclipse Mosquitto destaca por su simplicidad y eficiencia en entornos con recursos limitados. Su arquitectura monohilo garantiza estabilidad y facilidad de implementación, aunque limita su escalabilidad horizontal. A pesar de que su versión gratuita carece de funciones avanzadas, su bajo consumo de memoria y CPU lo convierte en una opción popular en entornos educativos y aplicaciones básicas de IoT [27].

Como se ha mencionado, Eclipse Mosquitto es un *broker* MQTT ligero y de código abierto, especialmente diseñado para dispositivos con restricciones de recursos —como sensores o microcontroladores—, donde la eficiencia energética y el bajo consumo son prioritarios. Al seguir el modelo *publish/subscribe*, facilita comunicaciones asíncronas, escalables y desacopladas, ideales para IoT. Su arquitectura incluye tres elementos fundamentales: el ejecutable del *broker* (mosquitto) y dos herramientas de línea de comandos: `mosquitto_pub` para publicar y `mosquitto_sub` para suscribirse, útiles para pruebas y automatización [28].

La configuración se centraliza en el archivo mosquitto.conf, un fichero de texto plano con sintaxis sencilla basada en pares clave-valor. Aunque opcional para ejecuciones mínimas, es indispensable para personalizar el comportamiento y reforzar la seguridad. Por defecto, Mosquitto escucha en el puerto 1883 (MQTT sobre TCP) y 9001 (WebSockets), pero admite múltiples *listeners*, lo que permite segmentar tráfico y aplicar políticas diferenciadas según la interfaz de red. La autenticación se gestiona mediante archivos de contraseñas y listas de control de acceso (ACL), con la opción *allow_anonymous* para habilitar o restringir conexiones sin credenciales. También soporta *plugins* de autenticación y cifrado TLS/SSL mediante certificados, esenciales en entornos productivos [29].

La persistencia, activable con la directiva *persistence*, permite almacenar en disco el estado de sesiones, suscripciones y mensajes pendientes (según el nivel de QoS), asegurando continuidad ante desconexiones frecuentes. La ubicación y nombre del archivo de persistencia pueden personalizarse mediante *persistence_location* y *persistence_file*. Para proteger la estabilidad del sistema, es posible establecer límites como número máximo de conexiones simultáneas, el tamaño de cola de mensajes o los nive-

les de QoS permitidos, evitando abusos y garantizando rendimiento bajo carga [30]. Finalmente, el sistema de *logs* —configurable en destino, formato, nivel de detalle y tipo de eventos— se convierte en una herramienta clave no solo para depuración y monitoreo, sino también para auditoría y cumplimiento de normativas de seguridad [31].

6.2. Cámaras Amcrest

La cámara de seguridad Amcrest 4MP ProHd Indoor WiFi IP Camera (modelo IP4M-1041W) es un dispositivo diseñado para vigilancia en interiores, brindando una combinación de características técnicas avanzadas y opciones de integración para desarrolladores. Este modelo proporciona una resolución de video de 4 megapíxeles a 30 fotogramas por segundo, garantizando imágenes claras y con alto nivel de detalle. Su campo de visión de 90° y capacidades de paneo de 355° e inclinación de 90°, permiten una cobertura amplia del área monitoreada. La conectividad Wi-Fi de 2.4 GHz facilita su instalación sin necesidad de utilizar cables Ethernet. Adicionalmente, cuenta con visión nocturna por medio de LEDs infrarrojos, audio bidireccional debido a su micrófono y altavoz integrados y opciones de almacenamiento tanto local, por medio de tarjetas microSD, como en la nube. La compatibilidad con protocolos ONVIF y RTSP permite su integración con múltiples sistemas de gestión de video [32].

Para desarrolladores interesados en la automatización y personalización, la cámara es compatible con la biblioteca de Python: python-amcrest. Esta herramienta permite interactuar con las cámaras Amcrest a través de su API: HTTP, ofreciendo funcionalidades como la obtención de información del dispositivo, capturas instantáneas, control de movimiento y manejo de eventos como detección de movimiento o pérdida de video. Además, el fabricante del producto brinda un SDK de API HTTP que permite acceder y configurar las cámaras. Este SDK proporciona una amplia gama de comandos CGI para controlar múltiples funciones de la cámara, como configuración de video, detección de movimiento y gestión de archivos [33].

6.2.1. Protocolos cámaras Amcrest

El protocolo RTSP (*Real-Time Streaming Protocol*) y el SDK de Python para cámaras Amcrest son herramientas ampliamente utilizadas en la transmisión y gestión de video en tiempo real, especialmente en sistemas de videovigilancia y monitoreo IP. RTSP es un protocolo de control de red diseñado para la transmisión de datos multimedia en tiempo real a través de redes IP. Permite a los clientes enviar comandos al servidor para iniciar, pausar, reanudar o detener la transmisión de video o audio, facilitando así el control remoto de sesiones de streaming [34]. Este protocolo es fundamental en cámaras de videovigilancia IP, como las de Amcrest, ya que permite establecer y gestionar flujos de video en vivo de manera eficiente [35]. RTSP opera bajo un modelo cliente-servidor, donde el cliente se conecta al servidor (por ejemplo,

una cámara IP) usando el puerto estándar 554. Posteriormente, envía solicitudes como *DESCRIBE*, *SETUP*, *PLAY*,*PAUSE* y *TEARDOWN*; recibe el flujo multimedia a través de canales separados, generalmente usando RTP (*Real-time Transport Protocol*). Este proceso ocurre en segundos, permitiendo una experiencia fluida para el usuario final [36].

El SDK de Python para Amcrest es un módulo compatible con Python 2.7 y 3.x que facilita la interacción de programación con cámaras Amcrest a través de su API HTTP. Permite realizar acciones como: consultar información del software de la cámara, capturar instantáneas, controlar el movimiento PTZ (*pan, tilt, zoom*) y grabar transmisiones en tiempo real en archivos locales [37]. Además, el SDK incluye utilidades de línea de comandos (amcrest-cli) y una interfaz de usuario en modo texto (TUI) para facilitar la configuración y manejo de múltiples cámaras.

Diversos estudios han evaluado el desempeño de RTSP en ambientes de transmisión de video en tiempo real, destacando su eficiencia y baja latencia en redes locales y de área amplia [36]. Sin embargo, RTSP puede enfrentar desafíos relacionados con la seguridad, como la transmisión de credenciales en texto claro y la falta de cifrado nativo, lo que requiere la implementación de capas adicionales de seguridad, por ejemplo, TLS o VPN, para proteger las transmisiones sensibles [38]. En cuanto al rendimiento, RTSP es adecuado para escenarios con múltiples clientes y altas demandas de calidad de video, aunque factores como la pérdida de paquetes y la latencia de red pueden afectar la experiencia del usuario [36].

6.3. Arquitecturas de red para laboratorios remotos

El diseño de un laboratorio remoto académico exige una arquitectura de red que equilibre múltiples requisitos: baja latencia para la interacción sincrónica, alta disponibilidad ante fallas, seguridad frente a amenazas externas y operabilidad sencilla para usuarios no especializados. Estos criterios han llevado a la adopción de diversas estrategias de conectividad, desde configuraciones locales tradicionales hasta soluciones híbridas basadas en la nube, edge computing y redes móviles. Esta sección presenta las alternativas técnicas más relevantes —NAT, VPN, despliegue local, la nube, conectividad 4G/5G— mostrando sus implicaciones en términos de rendimiento, seguridad y mantenimiento. Además, se ilustra la aplicación integrada de estos conceptos mediante una implementación representativa en el ámbito académico.

6.3.1. NAT y reenvío de puertos: simplicidad versus exposición

La traducción de direcciones de red (NAT) con reglas de *port forwarding* es una de las configuraciones más comunes para exponer servicios internos a Internet. Su principal ventaja radica en la facilidad de implementación, ya que solo requiere mapear puertos específicos en un router o firewall para redirigir tráfico externo hacia un servidor local. No obstante, este enfoque expone directamente los servicios internos

a la red pública, lo que representa un riesgo significativo si no se complementa con mecanismos adecuados de autenticación, cifrado y filtrado [39].

Desde la perspectiva de política de *firewalls*, el National Institute of Standards and Technology (NIST) recomienda aplicar el principio de *deny by default*: bloquear todo tráfico no explícitamente autorizado y abrir únicamente los puertos esenciales. Aunque el *port forwarding* es simple y de baja sobrecarga computacional, suele ser menos resistente frente a amenazas como escaneos automatizados, *exploits* conocidos o reconfiguraciones no intencionadas que un túnel VPN bien gestionado [40].

6.3.2. Redes privadas virtuales (VPN): seguridad cifrada y acceso controlado

Las redes privadas virtuales (VPN) ofrecen una alternativa más segura al NAT. Mediante túneles cifrados, las VPN integran a los usuarios remotos dentro de la red institucional, asignándoles una dirección IP interna y permitiéndoles acceder a servicios locales como si estuvieran físicamente presentes. Estudios recientes destacan que las VPN brindan mayor seguridad y flexibilidad, aunque requieren configuraciones más complejas y un sistema robusto de gestión de usuarios certificados [41].

Existen dos enfoques predominantes: IPsec y SSL/TLS. IPsec opera en la capa de red y constituye el estándar más consolidado para seguridad en la capa IP. Se recomienda especialmente cuando se implementa con módulos criptográficos validados bajo FIPS (Federal Information Processing Standards, estándares de seguridad criptográfica del NIST) y puede desplegarse en arquitecturas *gateway-to-gateway* o de acceso remoto, según los requisitos de autenticación, integridad y confidencialidad [42]. Por su parte, las SSL/TLS VPN operan en capas superiores (transporte o aplicación) y suelen implementarse en espacio de usuario, lo que puede ofrecer menor rendimiento que soluciones basadas en kernel como IPsec [43].

Esta diferencia es crítica en entornos donde la latencia y el *jitter* afectan la sincronicidad, como en la teleoperación de equipos o la transmisión de video en tiempo real. Por ello, NIST recomienda comparar ambas tecnologías considerando las aplicaciones involucradas, los clientes a conectar, los requisitos de acceso y modelar el impacto en latencia antes del despliegue definitivo [44].

6.3.3. Despliegue local, nube y edge computing: equilibrio entre latencia y escalabilidad

La elección entre servidores locales y servicios en la nube representa un eje fundamental en el diseño de laboratorios remotos. Los servidores locales proporcionan control total sobre el hardware y los datos, con menores latencias en redes internas. Sin embargo, implican una inversión inicial alta y costos continuos de mantenimiento eléctrico, de red y de seguridad física [45].

En contraste, las soluciones en la nube destacan por su escalabilidad, menor costo inicial y alta disponibilidad gracias a la redundancia geográfica. Sus desventajas incluyen posibles incrementos en la latencia si los servidores están en regiones lejanas y desafíos relacionados con privacidad y cumplimiento normativo [45]. Según la definición de NIST, la computación en la nube es el acceso bajo demanda a recursos compartidos y configurables —redes, cómputo, almacenamiento— con elasticidad y autoservicio, ofrecidos en modelos como IaaS o PaaS y despliegues público, privado o híbrido [46].

Para entornos sensibles a la latencia —como aquellos que involucran control de dispositivos físicos—, un enfoque híbrido con *edge computing* desplegado cerca del laboratorio (por ejemplo, en la red del campus o del ISP local) puede amortiguar las latencias críticas mientras se mantiene una observabilidad y gestión centralizadas [47].

6.3.4. Conectividad celular 4G/5G: redundancia y movilidad

La conectividad mediante SIMs 4G o 5G se presenta como una opción complementaria, especialmente valiosa en contextos con infraestructura de red fija inestable. Aporta redundancia y movilidad, permitiendo mantener el servicio activo incluso durante fallas en la red. Sus limitaciones incluyen el costo de los planes de datos de alta capacidad y la variabilidad en parámetros como latencia, *jitter* y pérdida de paquetes, que afectan aplicaciones sensibles como video en tiempo real o control remoto [48]. No obstante, la tecnología 5G mitiga muchas de estas limitaciones, permitiendo operaciones remotas confiables incluso en laboratorios educativos [48].

6.3.5. Ilustración práctica: estrategia de direccionamiento dual en entornos académicos

La aplicación coherente de los principios anteriores se observa en implementaciones académicas que combinan direccionamiento interno y externo para optimizar rendimiento y accesibilidad. Un caso representativo es el laboratorio remoto desarrollado en la Universidad del Quindío para la enseñanza de automatización y control, donde se adoptó una arquitectura con direccionamiento IP dual para garantizar la operación tanto local como remota de los sistemas [49].

En este esquema, el servidor (equipado con NGINX como servidor web ligero) opera simultáneamente con una dirección IP interna, asignada por la red del campus, y una dirección IP externa asociada a la red pública de Internet. La IP interna se emplea exclusivamente para la comunicación de baja latencia entre el servidor y los dispositivos físicos del laboratorio —tarjetas Texas Instruments LAUNCHXL-F28379D conectadas por USB, cámaras IP HiLook y una base de datos local—. Al no estar enrutada en Internet, esta configuración aísla los dispositivos críticos de accesos externos no autorizados y optimiza el tráfico dentro de la LAN institucional.

Por otro lado, la IP externa habilita la accesibilidad desde cualquier red pública mediante mecanismos de traducción de direcciones (NAT) y, en algunos casos, *port forwarding*, que permiten mapear solicitudes provenientes de Internet hacia el servidor local. Esta estrategia ilustra cómo la segmentación de tráfico —interno para control en tiempo real, externo para acceso remoto— permite conciliar los requisitos de seguridad, rendimiento y usabilidad en un entorno académico real [49]. No obstante, la dependencia exclusiva de NAT sin capas adicionales de seguridad puede resultar insuficiente en contextos con alta variabilidad de red, lo que refuerza la necesidad de complementar esta arquitectura con túneles cifrados o enlaces redundantes.

6.4. JSON Web Tokens como herramienta de seguridad

La autenticación es un pilar fundamental en páginas web y sistemas distribuidos, ya que, permite confirmar la identidad del usuario y controlar el acceso a recursos sensibles. En un entorno donde las páginas y aplicaciones web son accesibles desde diferentes dispositivos y ubicaciones, la autenticación segura es esencial para proteger la información y mantener la confianza del usuario. Debido a lo anterior, se han desarrollado múltiples métodos y protocolos para asegurar que el proceso de autenticación sea eficaz y fácil de implementar en arquitecturas modernas [50].

En este contexto, los JSON Web Tokens (JWT) se han consolidado como un mecanismo ligero y seguro para el intercambio de credenciales. Un JWT es un token compacto, basado en JSON, que contiene información codificada sobre la identidad del usuario y los permisos asociados. Lo distintivo de estos tokens es que son firmados criptográficamente, lo que garantiza que no pueden alterarse durante la transmisión. Esto permite que el servidor no tenga que mantener estado de sesión, facilitando la escalabilidad y eficiencia en sistemas distribuidos [51].

El uso de JWT en comparación con métodos tradicionales como OAuth y SAML, ofrece varias ventajas, tales como la reducción en la necesidad de consultas a base de datos para verificar sesiones y la posibilidad de integrar autenticación y autorización en una sola estructura. Sin embargo, es importante implementar JWT con buenas prácticas de seguridad como el uso de firmas seguras, expiración de tokens y protocolos HTTPS para proteger la confidencialidad y prevenir ataques. Considerando lo anterior, JWT se posiciona como un formato óptimo para la gestión segura de credenciales en aplicaciones web modernas [52].

6.4.1. Estructura de un JWT

La estructura del JSON Web Token está definida por el estándar RFC 7519, que especifica una manera compacta y autónoma para transmitir información segura entre partes. El JWT consta de tres componentes principales, que están codificados individualmente en BASE64 URL y separados por puntos. Estos componentes son el

Header, Payload y la Signature, conformando una estructura que facilita la autenticación y autorización en aplicaciones web [53].

El primer componente es el *header*, que consta de dos elementos esenciales: tipo de *token* (suele ser JWT) y el algoritmo criptográfico usado para firmar el token como HMAC SHA256 o RSA. Esta información se codifica en BASE64 URL para formar la primera parte del JWT. El *header* es clave para indicar cómo se debe verificar la autenticidad del token y asegura que el receptor pueda interceptar correctamente el contenido [54]. El *payload* es la segunda parte del *token* y contiene las afirmaciones sobre un usuario, como su identidad y permisos de acceso. Estas afirmaciones pueden ser registradas, públicas o privadas y aunque no estén cifradas, están firmadas evitar modificaciones no autorizadas. El *payload* también se codifica en `BASE64 URL` y su contenido debe manejarse con cuidado para no incluir información sensible sin protección adicional [55].

Finalmente, la *signature* es la tercera parte que garantiza la integridad del *token* y la autenticidad del emisor. Se genera usando el algoritmo especificado en el *header*, aplicando una función *hash:* sobre la concatenación codificada del *header* y el *payload* junto con una clave secreta. Esta firma digital evita que el *token* sea alterado durante la transmisión, siendo fundamental para la seguridad del proceso de autenticación [56].

6.4.2. Flujo de funcionamiento, limitaciones y desafíos de seguridad de JWT

El flujo de funcionamiento de un `JWT` comienza cuando un usuario se autentica por medio de sus credenciales ante el servidor. Si la autenticación es exitosa, el servidor genera un *token* o credencial firmada digitalmente, que incluye información del usuario y permisos. El cliente almacena este *token* y lo envía en cada petición posterior dentro del encabezado HTTP como un *Bearer Token*, permitiendo al servidor validar la firma para autorizar el acceso a los recursos solicitados [57]. El ciclo de vida de esta clave de acceso abarca tres etapas: emisión, expiración y renovación. La emisión ocurre después de la autenticación, la expiración inmediata limita el tiempo de validez del *token* para reducir riesgos y la renovación, mediante un refresh *token*, permite extender la sesión sin necesidad de volver a autenticar al usuario. Estas prácticas son esenciales para mantener un equilibrio entre usabilidad y seguridad en sistemas [58].

Existen limitaciones y desafíos de seguridad asociados al uso de `JWT`. Uno de los principales es la dificultad para revocar *tokens* antes de su expiración, lo que puede permitir accesos no autorizados si la credencial digital es comprometida. La exposición de la clave secreta utilizada para generar las firmas representa un riesgo grave, ya que un atacante podría generar accesos válidos. Entre los ataques más comunes están el robo de *tokens* y los *replay attacks*, donde un código válido es reutilizado maliciosamente [59].

Para mitigar estos riesgos se recomiendan buenas prácticas como el uso obligato-

rio de HTTPS para todas las comunicaciones, establecer tiempos de expiración cortos para las credenciales de acceso y utilizar *refresh tokens* en combinación para renovar sesiones. Además, la rotación de claves e implementación de mecanismos para la revocación de *tokens*, como listas negras, contribuyen a fortalecer la seguridad de la autenticación basada en JWT [60].

6.5. *Frameworks* Python para desarrollo web: Django y Flask (arquitectura, despliegue y servidores WSGI/ASGI)

El desarrollo de aplicaciones web modernas requiere de herramientas que integren seguridad, comunicación en tiempo real y despliegue eficiente en diversos entornos. En este panorama, Django y Flask se han consolidado como pilares del ecosistema Python, ofreciendo enfoques distintos en complejidad, modularidad y escalabilidad [61], [62]. Ambos se comunican con servidores web mediante interfaces estandarizadas: WSGI para entornos síncronos y ASGI para escenarios asincrónicos [63]. Esta sección presenta la arquitectura de ambos *frameworks* y explica los principios de WSGI y ASGI.

6.5.1. Arquitectura y funcionamiento de los *frameworks* Django y Flask

Django es un *framework* de alto nivel que sigue el principio *"batteries included"*, integrando funcionalidades como autenticación, administración y seguridad dentro de una arquitectura MTV (*Model–Template–View*) [62]. Este patrón separa claramente la lógica de datos, la presentación y el control, lo que favorece la mantenibilidad y escalabilidad. Al recibir una solicitud HTTP, Django la procesa mediante *middleware* y enrutamiento, ejecuta la vista correspondiente y genera una respuesta coherente usando modelos y plantillas [61]

Flask, en cambio, adopta una filosofía minimalista que prioriza la simplicidad y la flexibilidad. Ofrece solo lo esencial: enrutamiento, un servidor de desarrollo y el motor de plantillas Jinja2, todo ello sobre la base de Werkzeug, su gestor de solicitudes WSGI [64]. Esta ligereza lo hace ideal para microservicios, APIs RESTful o aplicaciones con restricciones de recursos. Además, mediante Blueprints permite organizar el código de forma modular [63].

Para abordar escenarios asincrónicos, Django incorpora Django Channels, una extensión que lo lleva más allá de WSGI hacia la compatibilidad con ASGI. Esta capa permite gestionar conexiones persistentes como WebSockets mediante "consumidores" asincrónicos y un sistema de *channel layers*. Así, se pueden construir interfaces interactivas, transmisiones en vivo o paneles colaborativos sin recargar la página. No obstante, esta potencia conlleva mayor complejidad en el escalado y mantenimiento [65].

6.5.2. Interfaces WSGI y ASGI: diferencias, servidores y despliegue en producción

WSGI (Web Server Gateway Interface) es el estándar clásico para ejecutar aplicaciones Python de forma síncrona. Diseñado para manejar una solicitud a la vez, prioriza la estabilidad y la integridad del procesamiento, lo que lo hace ideal para formularios, APIs transaccionales o paneles administrativos. Su simplicidad facilita la depuración y garantiza un flujo predecible de ejecución [61].

ASGI (Asynchronous Server Gateway Interface), por otro lado, es una evolución moderna que soporta concurrencia y protocolos como `WebSockets` o HTTP/2. Permite atender múltiples solicitudes simultáneamente dentro de un mismo hilo, reduciendo la latencia en aplicaciones que exigen interacción constante. Esta capacidad es esencial en contextos como *streaming* de video, chat en vivo o sistemas de teleoperación [63].

En producción, tanto `Django` como `Flask` se ejecutan mediante servidores especializados según el estándar elegido. Para WSGI, opciones como `Waitress`, `Gunicorn` o `uWSGI` traducen solicitudes HTTP en llamadas a la aplicación. `Waitress`, en particular, destaca por ser multiplataforma y completamente escrito en Python, ideal para entornos Windows o despliegues locales livianos [66].

Los servidores ASGI, como `Daphne` o `Uvicorn`, están diseñados para aplicaciones asincrónicas que mantienen conexiones abiertas y gestionan alta concurrencia. Son indispensables para aprovechar `Django Channels` o *frameworks* como FastAPI. Durante el despliegue, es habitual combinar estos servidores con NGINX o Apache como *reverse proxies*, que se encargan del tráfico estático, el balanceo de carga y el cifrado TLS [61].

CAPÍTULO 7

Diagnóstico del Robotat y análisis de protocolos de comunicación

Antes de proponer mejoras o rediseños en la infraestructura de un laboratorio de robótica, es fundamental comprender su estado actual, sus capacidades y sus limitaciones. Este capítulo presenta un diagnóstico detallado del Robotat, describiendo su arquitectura de red, los dispositivos involucrados y los protocolos de comunicación empleados al momento del análisis. A partir de esta caracterización, se realiza una evaluación comparativa entre los protocolos TCP y MQTT mediante un enfoque de Análisis de Decisión Multicriterio (MCDA), con el objetivo de identificar la alternativa más adecuada para habilitar una comunicación confiable, escalable y compatible con futuras extensiones del sistema.

7.1. Características del Robotat al momento del diagnóstico

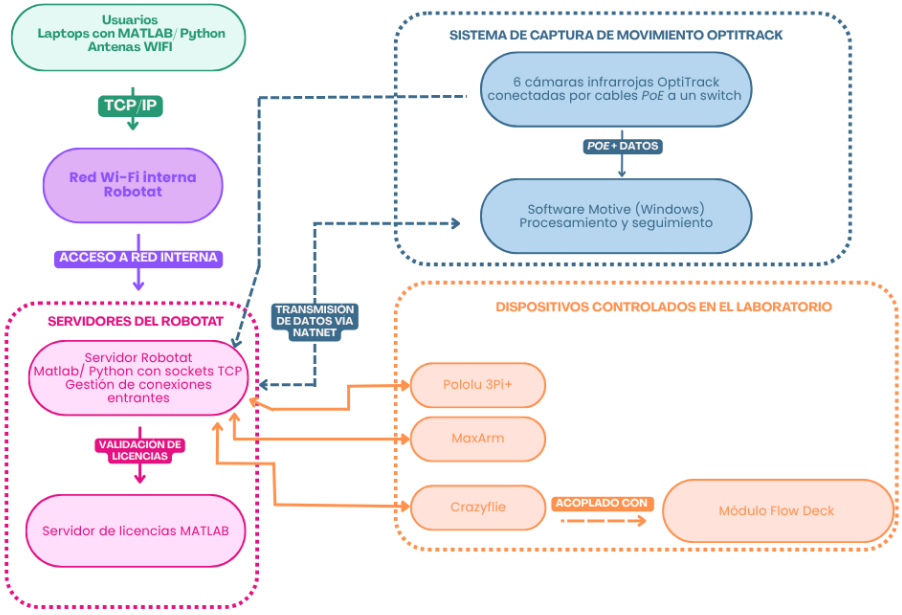
El laboratorio Robotat opera actualmente mediante una infraestructura de red local cerrada, diseñada para el control presencial de robots móviles, drones y un sistema de captura de movimiento. El acceso se realiza por medio de una red Wi-Fi interna, sin conexión a Internet y protegida por contraseña. Cada usuario emplea su computadora personal, equipada con MATLAB o Python y conectada mediante antenas Wi-Fi de alta ganancia, lo que permite el envío de comandos en tiempo real a través del protocolo TCP/IP.

En el núcleo de la arquitectura se ubica el servidor principal del Robotat, encargado de gestionar las conexiones entrantes mediante TCP y coordinar la comunicación con los dispositivos del laboratorio (Pololu 3Pi+, MaxArm y Crazyflie). Un servidor adicional administra las licencias de MATLAB. El sistema de captura de movimiento

OptiTrack, compuesto por seis cámaras infrarrojas conectadas mediante PoE (*Power over Ethernet)*, transmite datos procesados por el *software* `Motive` en Windows hacia el servidor Robotat a través del protocolo `NatNet`.

Esta configuración, ilustrada en la Figura 3, se limita al uso presencial dentro del laboratorio.

Figura 3. Arquitectura del Robotat al momento del diagnóstico.



Nota. El esquema muestra la red local interna del laboratorio, los servidores principales, los dispositivos controlados y el sistema de captura de movimiento `OptiTrack`, junto con los protocolos TCP/IP y `NatNet` empleados para la comunicación. Elaboración propia.

7.2. Ventajas vs desventajas arquitectura previa Robotat

Cuadro 1. Ventajas y desventajas del funcionamiento actual del Robotat

Ventajas	**Desventajas**
La red local cerrada proporciona un entorno seguro y aislado, libre de interferencias externas.	El sistema solo es accesible de forma presencial, lo que limita su uso a usuarios dentro del campus.
El uso de direcciones IP fijas facilita la identificación y conexión directa con cada dispositivo del laboratorio.	No existe acceso remoto ni integración con servicios en la nube, lo que restringe la escalabilidad y flexibilidad de la plataforma.
La comunicación mediante el protocolo TCP/IP asegura estabilidad y confiabilidad en el control de los dispositivos.	El servidor principal debe ejecutarse en Windows, ya que el software `Motive` no es compatible con Linux, lo que obliga a usar múltiples equipos.
El software `Motive` permite crear cuerpos rígidos a partir de *markers*, con identificación única, lo cual estandariza la captura de movimiento.	`Motive` solo ofrece visualización básica en escala de grises, sin interfaz gráfica integrada que unifique el control de robots y cámaras.
El switch PoE simplifica la conexión de cámaras, al proveer tanto datos como energía mediante un solo cable.	La gestión de conexiones simultáneas (NatNet, Python, MATLAB) puede generar conflictos y no permite ejecutar de forma paralela el modo Robotat y Linux.
La arquitectura funciona sin depender de servicios externos, adecuada para sesiones prácticas controladas en campus.	La infraestructura requiere antenas USB Wi-Fi de alta ganancia para cada usuario, lo que incrementa el costo y la complejidad de instalación.

Nota. Este análisis resume las fortalezas y limitaciones del Robotat, destacando la seguridad y confiabilidad de su red local frente a sus restricciones de accesibilidad; sirve como base para proponer una infraestructura más flexible y accesible. Elaboración propia

7.3. Análisis de Decisión Multicriterio

El Análisis de Decisión Multicriterio (MCDA, por sus siglas en inglés) es una metodología que permite evaluar y comparar múltiples alternativas tomando en cuenta diversos criterios, facilitando decisiones más informadas y equilibradas. Esta técnica resulta especialmente valiosa en situaciones donde las decisiones deben considerar varios objetivos que, en ocasiones, pueden ser contradictorios. Ámbitos como la gestión de recursos, la planificación estratégica o la elección de tecnologías se benefician especialmente de ella. MCDA brinda un enfoque sistemático que permite a los res-

ponsables de tomar decisiones identificar los criterios más relevantes, establecer su importancia relativa mediante la asignación de pesos y analizar el desempeño de cada alternativa frente a dichos criterios [67].

Una de las fortalezas de este método es su capacidad de integrar tanto datos cuantitativos como cualitativos, permitiendo una evaluación más integral de las opciones disponibles. Por ejemplo, en el ámbito de la salud, se ha utilizado este análisis para evaluar tecnologías médicas tomando en cuenta la eficacia clínica, el costo, equidad y aceptabilidad social [68]. Asimismo, otros métodos de decisión son empleados dentro del MCDA para facilitar la comparación y selección de alternativas [69].

El enfoque MCDA se aplicó para comparar los protocolos de comunicación MQTT y TCP según criterios pertinentes a la operación del Robotat. El objetivo fue fundamentar la selección del protocolo más adecuado para la infraestructura del laboratorio.

7.4. Evaluación multicriterio (MCDA) de protocolos de comunicación: MQTT vs TCP.

Cuadro 2. MCDA: Facilidad, soporte y comunicación

Criterio	**Peso (%)**	**MQTT (0–10)**	**TCP (0–10)**
Facilidad de implementación	15	9	6
Soporte multiusuario	20	10	6
Comunicación bidireccional	15	10	9

Nota. En este cuadro se muestra la comparación de criterios de evaluación MCDA para MQTT y TCP. Elaboración propia.

Cuadro 3. MCDA: Historial, escalabilidad y eficiencia.

Criterio	**Peso (%)**	**MQTT (0–10)**	**TCP (0–10)**
Registro de historial	10	9	8
Escalabilidad	10	9	6
Latencia y eficiencia local	10	9	9

Nota. El cuadro resume la valoración comparativa (0–10) de MQTT y TCP en criterios clave de desempeño local. Elaboración propia.

Cuadro 4. MCDA: Seguridad, compatibilidad y puntaje total.

Criterio	**Peso (%)**	**MQTT (0–10)**	**TCP (0–10)**
Seguridad en red cerrada	5	8	8
Compatibilidad futura (IoT, nube)	5	10	6
Fiabilidad y persistencia	10	9	8
Total ponderado (0–10)		**9.4**	**7.2**

Nota. El cuadro muestra la evaluación multicriterio (MCDA) de MQTT y TCP en aspectos de seguridad, compatibilidad y fiabilidad, incluyendo el puntaje total ponderado. Elaboración propia.

Cuadro 5. MCDA. Puntajes ponderados por criterio y totales.

Criterio	**MQTT (ponderado)**	**TCP (ponderado)**
Facilidad de implementación	1.35	0.90
Soporte multiusuario	2.00	1.20
Comunicación bidireccional	1.50	1.35
Registro de historial	0.90	0.80
Escalabilidad	0.90	0.60
Latencia y eficiencia local	0.90	0.90
Seguridad en red cerrada	0.40	0.40
Compatibilidad futura (IoT/nube)	0.50	0.30
Fiabilidad y persistencia	0.90	0.80
Total ponderado (0–10)	$9.35 \approx 9.4$	$7.25 \approx 7.4$

Nota. Se muestran los puntajes ponderados por criterio y el total de la comparación MCDA entre MQTT y TCP. Elaboración propia.

7.4.1. Justificación

MQTT obtuvo una puntuación más alta debido a su facilidad de implementación mediante bibliotecas especializadas, mientras que TCP requirió programación manual y manejo de múltiples *sockets*. En las pruebas realizadas, MQTT permitió una comunicación fluida y bidireccional entre múltiples clientes sin necesidad de ejecutar varias instancias del código, algo que en TCP se logró con esfuerzo adicional. Además, MQTT gestionó eficientemente el registro de mensajes utilizando *callbacks*. Ambos protocolos fueron eficientes en redes locales cerradas, pero MQTT ofreció un mejor soporte para persistencia y compatibilidad futura gracias a características como QoS y retención de mensajes. Por estas razones, se recomienda MQTT como el protocolo más adecuado para la infraestructura de comunicación del Robotat.

7.4.2. Asignación de puntajes

La asignación de puntajes en la evaluación multicriterio (MCDA) se realizó de manera empírica, combinando dos fuentes: las características técnicas de cada protocolo y las pruebas prácticas efectuadas en el Robotat. En primer lugar, se definieron los criterios y se les asignó un peso porcentual de acuerdo con su relevancia en la infraestructura del laboratorio. Por ejemplo, se dio mayor peso al soporte multiusuario y a la facilidad de implementación, ya que el Robotat está orientado a sesiones con varios estudiantes conectados simultáneamente y el tiempo disponible para programar es limitado.

Posteriormente, cada protocolo fue calificado en una escala de 0 a 10 con base en su desempeño frente a cada criterio. Así, MQTT recibió puntajes altos en facilidad de implementación y soporte multiusuario debido a la existencia de bibliotecas (como `paho-mqtt`) y a su arquitectura *publish/subscribe* que simplifica la conexión de múltiples clientes. En contraste, TCP obtuvo puntajes menores porque requiere programación manual de *sockets* y un manejo más complejo de múltiples conexiones. En escalabilidad y compatibilidad futura con IoT o servicios en la nube, MQTT también superó a TCP por estar diseñado precisamente para entornos distribuidos. Por su parte, en latencia y seguridad dentro de una red cerrada ambos protocolos mostraron rendimientos equivalentes, motivo por el cual recibieron puntajes similares en esos criterios.

Finalmente, se multiplicaron los puntajes por sus respectivos pesos y se sumaron para obtener el total ponderado. De esta forma, MQTT alcanzó un valor de **9.4**, mientras que TCP obtuvo **7.3**, confirmando que MQTT representa la mejor opción de comunicación para la infraestructura del Robotat.

7.4.3. Pruebas prácticas realizadas

Además del análisis comparativo teórico, se desarrollaron pruebas iniciales de comunicación tanto con *sockets* TCP como con el protocolo MQTT. Para ello se implementaron programas en Python que permitieron evaluar el envío y recepción de mensajes en condiciones similares a las del Robotat.

En el caso de TCP, se diseñó un servidor multicliente basado en la biblioteca estándar *socket* y un cliente asociado que intercambiaba mensajes en tiempo real mediante hilos de ejecución (*threading*). Estas pruebas mostraron que, aunque la comunicación era confiable y de baja latencia, la gestión de múltiples conexiones simultáneas requería mayor complejidad en la programación y control de estados.

En paralelo, se implementó un cliente MQTT utilizando la biblioteca `paho-mqtt`, configurado para conectarse a un *broker* Mosquitto. En este caso, se validó el uso de temas de publicación/suscripción (*topics*) y la persistencia de mensajes en formato `JSON` con marca de tiempo, usuario y contenido. El manejo de múltiples usuarios se realizó de forma nativa a través del *broker*, simplificando significativamente la lógica

de programación respecto a la alternativa con TCP.

Si bien los códigos empleados constituyen prototipos iniciales, los resultados experimentales respaldaron el análisis multicriterio, evidenciando que MQTT ofrece una solución más escalable, flexible y adecuada para la futura integración de los servicios del Robotat.

7.4.4. Conclusión

De acuerdo con los resultados del análisis MCDA, MQTT obtuvo un puntaje ponderado de 9.4 frente a 7.3 de TCP. Esto confirma que MQTT es la mejor opción para el sistema de comunicación del proyecto, al ofrecer soporte nativo para múltiples clientes, facilidad de expansión y herramientas modernas que facilitan su integración en entornos distribuidos.

CAPÍTULO 8

Reestructuración de la infraestructura de red local del Robotat

El diagnóstico previo mostró que la infraestructura de red del Robotat, aunque funcional para sesiones presenciales, presenta limitaciones importantes en escalabilidad, accesibilidad y flexibilidad. A partir de los hallazgos encontrados —y considerando la evaluación multicriterio que posicionó al protocolo MQTT como superior a TCP— surgió la necesidad de consolidar los servicios del laboratorio en un servidor dedicado.

A continuación, se describe el proceso de reestructuración de la red local del Robotat mediante dicha consolidación. Dado el carácter iterativo del desarrollo, la metodología y los resultados se presentan de forma integrada, avanzando desde pruebas conceptuales con datos simulados hasta la integración completa con el sistema real.

8.1. Pruebas iniciales con datos simulados en entorno local

Siguiendo la arquitectura definida en el Capítulo 7, que establece a MQTT como protocolo central de comunicación, se implementó un programa en Python para simular el flujo de datos que posteriormente se integraría con el sistema real de captura de movimiento. En este código se utilizaron las bibliotecas `numpy` y `random` para generar valores aleatorios que imitaban posiciones y orientaciones. Además, se empleó `datetime` para agregar marcas de tiempo y `json` para estructurar la información en paquetes estandarizados. Para la comunicación con el *broker*, se usó la biblioteca `paho-mqtt`, que gestionó tanto la conexión como la publicación de los mensajes.

En esta etapa inicial, el *broker* Mosquitto se configuró en el mismo equipo de desarrollo (*localhost*). Esto permitió reproducir un escenario de transmisión y recepción de datos sin necesidad de *hardware* externo. Así, se validó la lógica básica del

flujo: creación de paquetes `JSON` publicación en un tópico definido y recepción por parte de uno o varios clientes suscritos. El *broker* gestionó correctamente múltiples publicaciones consecutivas, sin pérdidas ni duplicaciones. Este resultado demostró que la infraestructura de mensajería soportaba la persistencia de datos en tiempo real, incluso bajo generación aleatoria intensiva.

8.2. Migración al servidor del Robotat

Tras la validación inicial, las pruebas se trasladaron al servidor del Robotat, manteniendo datos simulados pero enviados a través de la red local hacia un *broker* Mosquitto instalado en el servidor del laboratorio (Windows). Se configuró el archivo `mosquitto.conf` para definir la dirección IP, permitir conexiones anónimas, habilitar persistencia y especificar la ruta de almacenamiento. Esta etapa confirmó la compatibilidad del *broker* con la infraestructura existente y su capacidad para gestionar múltiples publicaciones sin pérdidas, incluso bajo carga intensiva, demostrando estabilidad en un entorno real.

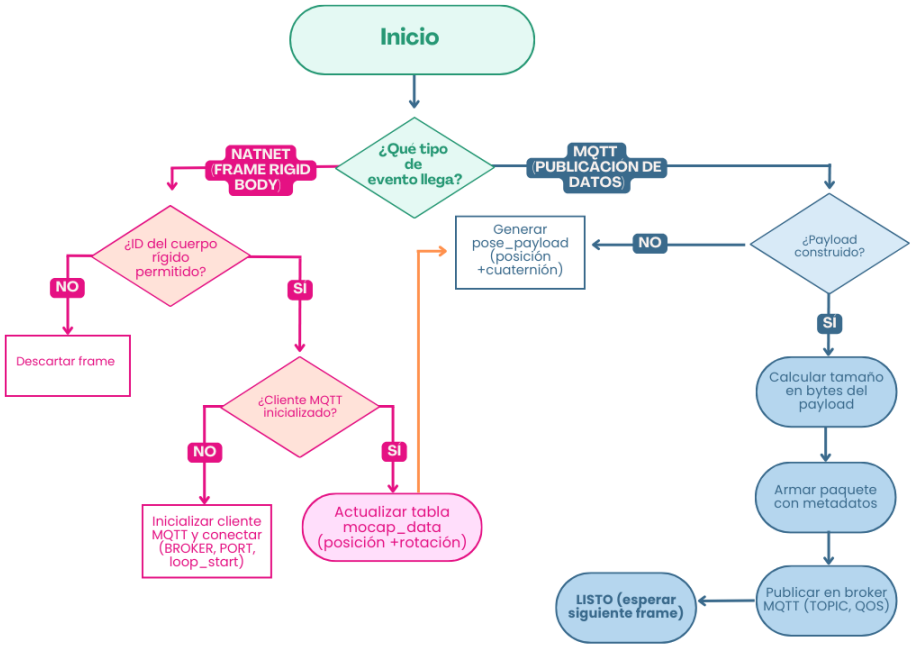
8.3. Integración con datos reales del sistema de captura de movimiento

La fase final integró el flujo real de datos del sistema `OptiTrack` mediante la biblioteca `NatNetClient`, conectando directamente con el *software* `Motive`. La publicación de los mensajes se realizó con la librería `paho-mqtt`, configurando la dirección IP, el puerto y el tópico `mocap/all`. Para estandarizar la comunicación, se definieron enumeraciones que codifican de forma uniforme el origen, el tipo de mensaje y el identificador del cuerpo rígido, facilitando la integración futura con otros dispositivos del Robotat. Cada marco recibido se encapsuló en un paquete con metadatos esenciales —incluyendo marca de tiempo, tamaño del *payload* y un campo reservado para *checksum*. El formato del paquete era el siguiente:

```
    packet = {
        "src": CURRENT_SOURCE.value,
        "pts": datetime.utcnow().timestamp(),
        "ptp": PacketType.MOCAP.value,
        "pid": int(id),
        "psb": payload_size_bytes,
        "pld": pose_payload,
        "cks": None,
    }
```

Los paquetes se publicaban en el tópico `mocap/all`, permitiendo recepción simultánea en múltiples clientes suscritos. Durante las pruebas, la transmisión se mantuvo consistente, sin pérdidas ni duplicados, validando la solidez del esquema MQTT frente a la lógica basada en *sockets* TCP utilizada anteriormente. El diseño resultante estandariza la distribución de datos en la red, asegurando trazabilidad, replicabilidad y monitoreo eficaz del flujo de comunicación; este se ilustra en la Figura 4.

Figura 4. Flujo de ejecución del código Publisher MQTT + NatNet.



Nota. Flujo de ejecución del programa MQTT utilizado en el sistema. Elaboración propia.

8.4. Análisis general de resultados

El conjunto de pruebas confirmó que MQTT simplifica significativamente la gestión de múltiples clientes, al delegar en el *broker* funciones que, con TCP, exigían mecanismos adicionales de control. La arquitectura demostró flexibilidad: la transición de datos simulados a reales se logró sin alterar la lógica central, lo que facilita la integración con diversas fuentes. Las publicaciones se mantuvieron estables y sin pérdidas, cumpliendo con los requisitos de fiabilidad para aplicaciones en tiempo real.

Además, la estandarización del paquete —mediante enumeraciones para metadatos y un campo reservado para *checksum*— permitió verificar la integridad de los mensajes y garantizar una interpretación uniforme entre clientes. El sistema operó de forma consistente tanto en entornos locales como en el servidor del Robotat, validando su compatibilidad con la infraestructura actual y su preparación para una futura migración.

CAPÍTULO 9

Diseño e implementación del sistema de autenticación del Robotat

La seguridad en el acceso constituye un requisito indispensable para el correcto funcionamiento de Robotat, un laboratorio que integra recursos físicos como robots móviles, brazos manipuladores y cámaras de video en tiempo real. Si bien la red local ofrece cierta protección al estar aislada de Internet, fue necesario desarrollar un sistema de autenticación capaz no solo de controlar quién accede al entorno, sino también de asignar niveles diferenciados de interacción según el rol de cada usuario. Esta necesidad cobra mayor relevancia ante la meta futura de exponer la interfaz web de Robotat a Internet, con el fin de permitir el control remoto de los dispositivos del laboratorio. A continuación se presenta el proceso de diseño e implementación de dicho sistema, incluyendo la metodología seguida y los pasos concretos que condujeron a su despliegue. Asimismo, se analizan los resultados obtenidos, evaluando su impacto en términos de organización, seguridad y experiencia de uso.

9.1. Arquitectura del *backend* y seguridad en la autenticación

En el *backend* se desarrolló la lógica central de autenticación utilizando `Django` y `Django REST Framework (DRF)`, estableciendo una base escalable para la gestión de usuarios. La metodología adoptada incluyó el uso de *tokens* `JWT (JSON Web Tokens)` como mecanismo de validación de sesiones. Cada inicio de sesión genera un *token* con validez limitada, que debe incluirse en todas las solicitudes a recursos protegidos. Estos *tokens* incorporan *claims* personalizados —como rol, identificador de usuario y tiempo de expiración—, lo que permitió tomar decisiones basadas en permisos directamente desde el *token*, sin consultar la base de datos en cada petición. Esta estrategia redujo la latencia y mejoró el rendimiento general del sistema.

Para manejar *tokens* comprometidos o caducados, se integró una lista de revocación (*blacklist*) que invalida *tokens* al cerrar sesión o al cambiar la contraseña [70]. Esta medida evitó la reutilización no autorizada de credenciales y facilitó el control de sesiones activas. Paralelamente, el módulo de cambio y recuperación de contraseñas reforzó la seguridad mediante el cifrado de credenciales con `PBKDF2` y *salt* aleatorio [71], transmitiéndolas exclusivamente por `HTTPS`. Durante las pruebas, cada actualización de contraseña invalidó automáticamente todos los *tokens* vigentes, en concordancia con las buenas prácticas de seguridad.

Se implementó además un sistema de *logs* de auditoría para registrar accesos exitosos, fallidos y modificaciones sensibles [72]. Cada evento incluyó el *token* asociado y una marca de tiempo, lo que permitió rastrear con precisión las acciones de los usuarios. Este mecanismo resultó especialmente útil durante las pruebas del microservicio y los ajustes de permisos, al facilitar la identificación de solicitudes legítimas o erróneas.

Adicionalmente, se incorporó `Django Channels` para habilitar comunicación en tiempo real mediante `WebSockets`. Este componente gestionó la actualización instantánea de estados del sistema y mensajes de servicio. El despliegue se realizó con `Daphne`, un servidor `ASGI` diseñado para manejar simultáneamente `WebSockets` y peticiones `HTTP`. Esta configuración permitió transmitir eventos sin interrupciones y sin interferir con la autenticación basada en `JWT`.

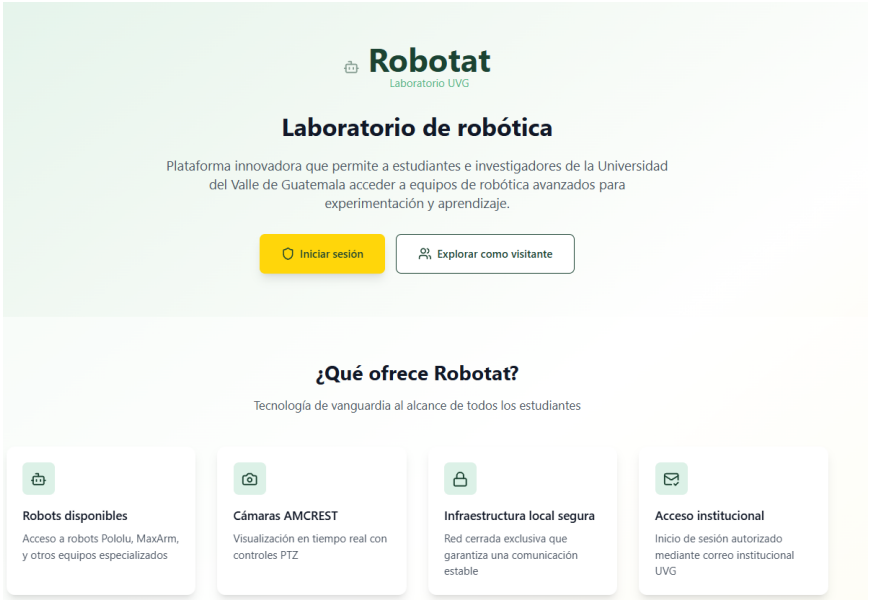
Finalmente, se implementó un mecanismo de refresco de *tokens* que extendió sesiones activas sin requerir nuevas credenciales, garantizando continuidad sin comprometer la seguridad. En conjunto, los resultados demostraron que la combinación de autenticación basada en `JWT`, gestión segura de sesiones, auditoría detallada y comunicación en tiempo real mediante `Channels` y `Daphne` conforma un *backend* seguro, modular y eficiente, capaz de proteger los recursos del sistema mientras mantiene una experiencia fluida para el usuario.

9.2. Arquitectura del *frontend* y experiencia de usuario

En el *frontend*, la herramienta de autenticación se construyó sobre `React`, lo que permitió organizar la interfaz en componentes reutilizables y mantener un control eficiente del estado de la aplicación. El uso de `Vite` como *bundler:* facilitó tiempos de compilación reducidos y una carga inicial más rápida, aspecto fundamental para mejorar la experiencia de usuario en un sistema que será utilizado de forma recurrente por diferentes perfiles [73]. Para el diseño visual se optó por `Tailwind CSS`, que aportó un sistema de clases utilitarias capaz de mantener coherencia en los estilos, con una estética moderna y adaptable a distintos tamaños de pantalla. Además, la plataforma cuenta con modo día y noche, configurable por cualquier usuario. Es importante resaltar que solo se incluyeron algunas imágenes representativas en la sección de resultados, las demás vistas de la página web podrán encontrarse en el capítulo 15.

La página de inicio (Figura 5) ofrece una vista general del sistema y enlaces contextuales según el estado de autenticación del usuario. La vista de inicio de sesión (Figura 6) incluye validaciones en tiempo real. Estas muestran mensajes claros cuando las credenciales son incorrectas o cuando el dominio del correo no es el institucional, evitando así intentos de acceso no autorizados. La misma interfaz incluye un flujo integrado para el cambio de contraseña, accesible desde el formulario de *login*.

Figura 5. Página de inicio de la plataforma del Robotat.



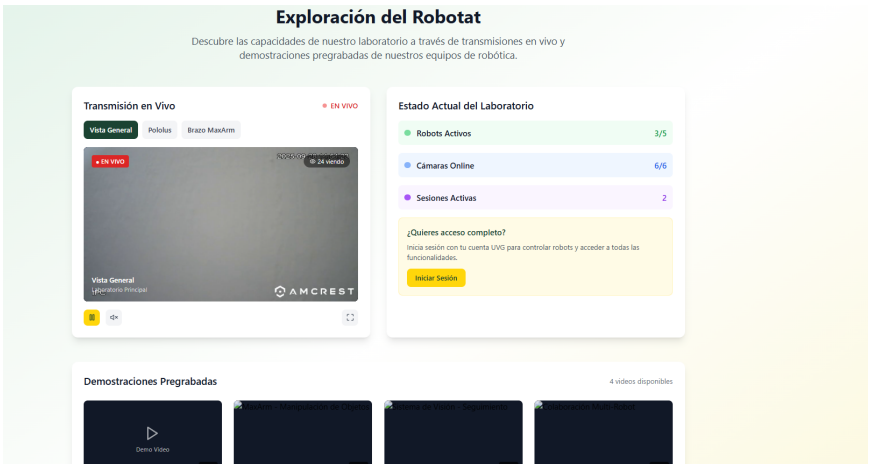
Nota. Página de inicio que introduce al usuario al sistema y permite el acceso a las funciones principales. Elaboración propia.

Figura 6. Página de autenticación y módulo de recuperación de credenciales.

Nota. Interfaz de acceso que une las funciones de *login* y gestión de contraseña. Elaboración propia.

Una vez autenticado, el *frontend:* interpreta el rol contenido en el token `JWT` y redirige automáticamente al usuario hacia la interfaz correspondiente: panel administrativo, área de docente, entorno de estudiante, espacio de investigador o vista de visitante [23]. Los visitantes no autenticados acceden a una interfaz limitada (Figura 7), que permite explorar información pública sin comprometer la seguridad del sistema.

Figura 7. Vista de visitante.

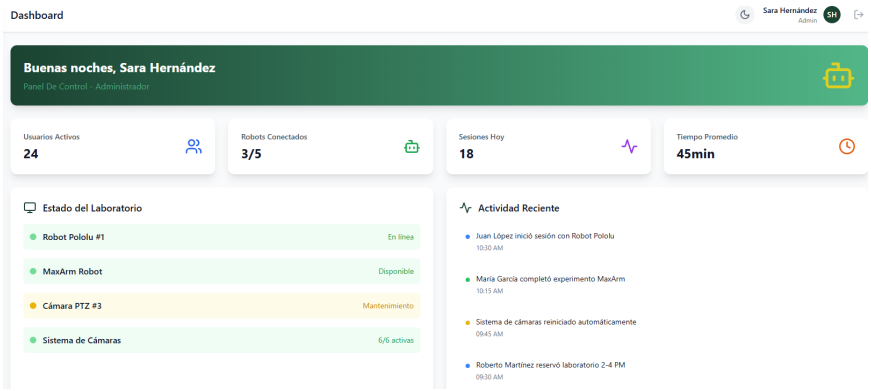


Nota. La vista de visitante permite el acceso restringido a la plataforma, mostrando únicamente información general sin requerir autenticación. Elaboración propia.

9.2.1. Interfaz del administrador

La interfaz del administrador representa el nivel más completo y funcional de la plataforma, ya que integra en un solo entorno las herramientas para: supervisión, control y gestión integral del Robotat. Al iniciar sesión, el administrador accede a un panel centralizado (Figura 8) que presenta indicadores clave en tiempo real: número de usuarios activos, robots conectados, sesiones ejecutadas durante el día, actividad reciente en la plataforma y tiempo promedio de uso de los recursos.

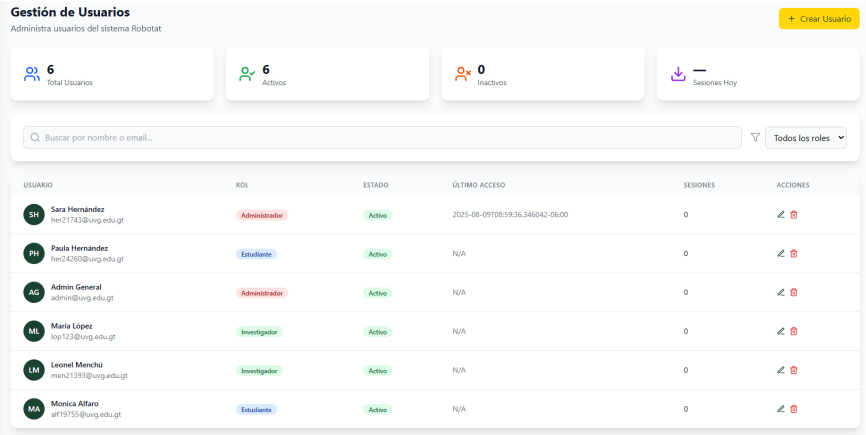
Figura 8. Panel principal del administador.



Nota. Esta visión inmediata permite evaluar de un vistazo el estado operativo del sistema. Elaboración propia.

Desde la misma interfaz, se puede acceder al módulo de gestión de usuarios (Figura 9), donde es posible filtrar perfiles por rol, crear nuevas cuentas, editar información existente y habilitar o deshabilitar accesos. Esta funcionalidad garantiza un control flexible, seguro y escalable sobre la comunidad de usuarios. En la sección dedicada al control del laboratorio, se muestra el estado detallado de cada robot, incluyendo la fecha y hora del último *ping*, su disponibilidad para nuevos experimentos y cualquier alerta relacionada con la conectividad.

Figura 9. Página de gestión de usuarios.

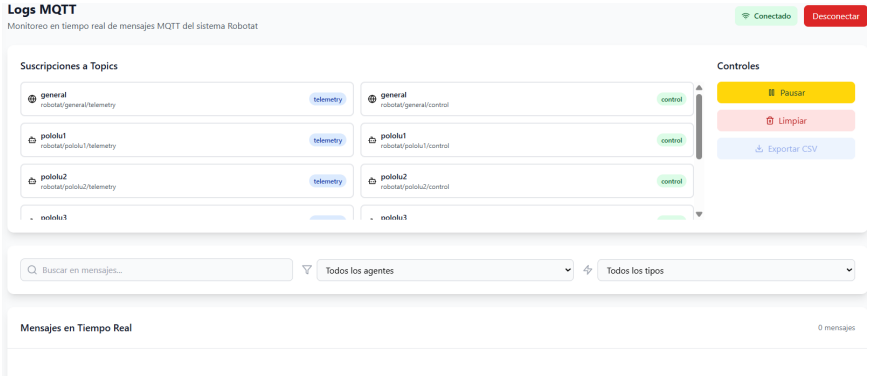


Nota. Vista de gestión para crear, modificar y eliminar usuarios del sistema. Elaboración propia.

El administrador, además, dispone de un módulo de monitoreo de cámaras que no solo permite visualizar transmisiones en vivo, sino también grabarlas y manipularlas mediante controles PTZ. Esto amplía significativamente las capacidades de observación y documentación en sesiones experimentales. El apartado de historial recopila información detallada sobre las interacciones en la plataforma: quiénes se conectaron, los momentos de mayor concurrencia y qué usuarios permanecieron activos durante períodos prolongados. Este registro constituye una fuente valiosa para la trazabilidad del sistema.

Adicionalmente, se cuenta con una pestaña de Logs MQTT y puede observarse en la Figura 10. En esta página el usuario puede elegir a que tópico suscribirse y obtener información que se está transmitiendo desde `OptiTrack`. Los tópicos disponibles se dividen en número de agente y posteriormente, en telemetría o control. Además, esta pestaña cuenta con una función para exportar los datos mostrados en la consola a un archivo csv. Es imporante resaltar que esta vista es exactamente la misma para todos los usuarios sin importar su rol.

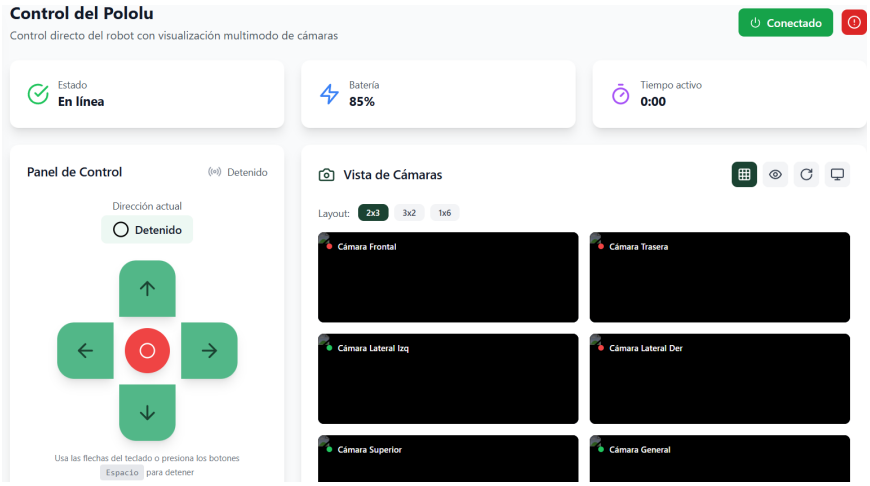
Figura 10. Página de suscripción a los tópicos MQTT.



Nota. Vista de la consola MQTT con registro de mensajes y suscripción a tópicos. Elaboración propia.

Asimismo, se incorporó una página dedicada al control del robot Pololu 3Pi+ (ver Figura 11). La interfaz permite dirigir el robot mediante botones en pantalla o usando las flechas del teclado, creando un control intuitivo y accesible. Además, se integró un panel de estadísticas que muestra información como el tiempo total de uso y el historial reciente de acciones, permitiendo al administrador supervisar la actividad del robot y su disponibilidad. Todo el acceso está protegido mediante autenticación con `JWT` y validación de permisos por rol. Para que el usuario pueda controlar el robot y ver en tiempo real sus acciones, se agregó un modal de visualización de las cámaras Amcrest; este es personalizable dependiendo de los gustos del usuario.

Figura 11. Control web del Pololu 3Pi+.

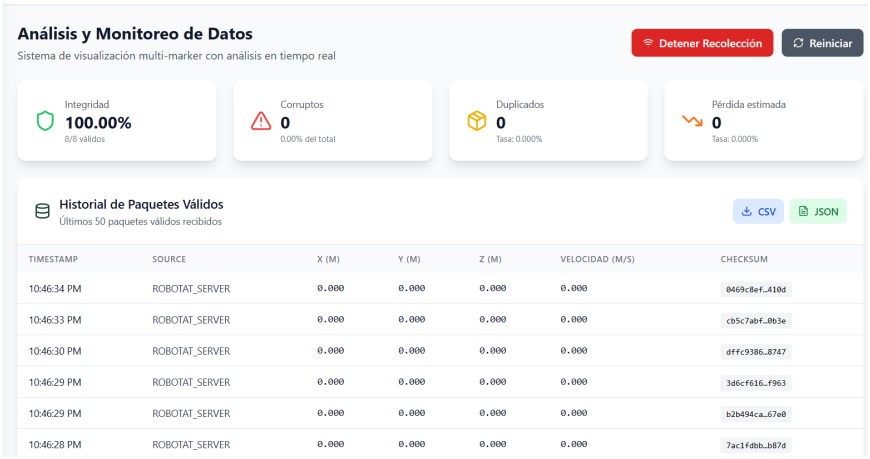


Nota. Módulo que permite controlar el Pololu 3Pi+ mediante flechas. Elaboración propia.

Se añadió también una página de análisis y monitoreo que procesa en tiempo real

los datos del sistema de captura (Figura 12). La interfaz visualiza *markers*, valida la integridad de los paquetes mediante *checksum* e identifica origen, marca de tiempo, tipo e identificador de cada mensaje. Además, genera gráficas automáticas de trayectoria (x, y,z), eventos y cambios de estado. Esta herramienta permitió confirmar la estructura de los datos transmitidos y detectar paquetes perdidos, duplicados o corruptos.

Figura 12. Control web del Pololu 3Pi+.



Nota. Módulo que permite controlar el Pololu 3Pi+ mediante flechas. Elaboración propia.

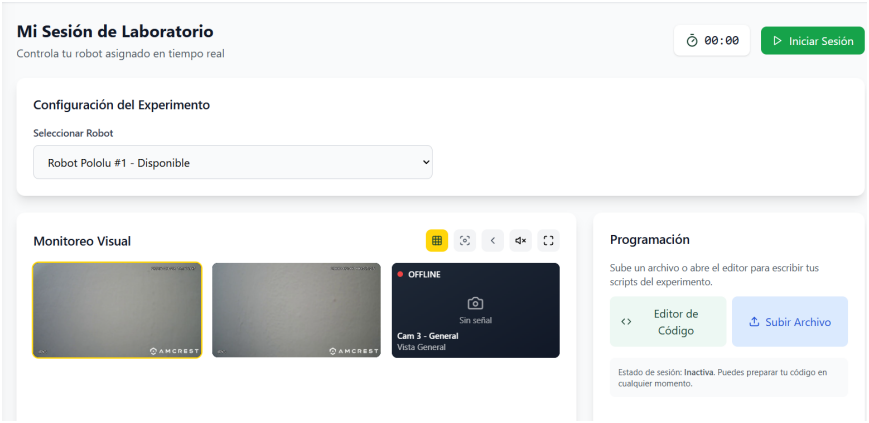
9.2.2. Interfaz del estudiante

La interfaz de estudiante fue diseñada con un enfoque pedagógico; se priorizó la claridad en la presentación de la información y el acceso inmediato a las funciones necesarias para realizar las prácticas de laboratorio. Al ingresar, el usuario accede a un panel personalizado que muestra estadísticas sobre su participación y desempeño en la plataforma. Ese *dashboard* incluye datos como el tiempo total de uso, el número de experimentos realizados, los resultados obtenidos y el estado actual de los recursos del laboratorio. De forma clara accesible, se indica, por ejemplo, si un robot Pololu 3Pi+ está en línea, si el manipulador MaxArm está disponible o en mantenimiento y cuál es la condición de las cámaras Amcrest. Además, el panel presenta una sección de actividad reciente, que permite a los estudiantes revisar sus interacciones más recientes y planificar mejor sus próximas sesiones.

En la sección "Mi sesión", el usuario dispone de un entorno centralizado para ejecutar experimentos. Allí, puede seleccionar el robot a utilizar, cargar el código correspondiente y acceder a la transmisión en vivo de las cámaras (Figura 13). La página cuenta con distintos modos de visualización para las cámaras: carrusel, cuadrícula o pantalla completa. Este diseño se adapta tanto a quienes necesitan monitorear varias perspectivas simultáneamente como a quienes prefieren enfocarse en un detalle específico. La interfaz también incluye una herramienta para marcar momentos clave

durante la ejecución, permitiendo generar anotaciones o referencias temporales. Estos marcadores se exportan en formato csv, facilitando su análisis posterior o su inclusión en reportes académicos.

Figura 13. Visualización de cámaras modo estudiante.



Nota. Página de cámaras en tiempo real con permisos limitados para estudiantes. Elaboración propia.

La sección "Resultados" actúa como un repositorio organizado de los experimentos ya realizados. Cada registro contiene información detallada: nombre del experimento, estado de éxito, duración, fecha de ejecución y los marcadores creados en tiempo real. Además, se ofrecen indicadores globales como el número de experimentos exitosos, el tiempo total invertido y un porcentaje promedio de efectividad. Los resultados pueden descargarse en distintos formatos —`.csv`, `.mat` o `.json`—, lo que amplía las posibilidades de análisis según la herramienta elegida por el estudiante. Este módulo no solo sistematiza la experiencia práctica, sino que también fomenta la autoevaluación y la mejora continua mediante métricas de desempeño individual.

El apartado "Material de apoyo" reúne documentos PDF, ejemplos de código y videos explicativos que guían al estudiante en la preparación y ejecución de sus prácticas. Este espacio complementa tanto el aprendizaje teórico como el práctico, ya que ofrece recursos diseñados para reforzar el estudio autónomo y estandarizar las metodologías de trabajo. Finalmente, la interfaz incluye un módulo de *logs* MQTT, igual al de administradores e investigadores. En este espacio, los estudiantes pueden observar en tiempo real los mensajes intercambiados entre el *broker* y los dispositivos, lo que les brinda una visión práctica de la comunicación en sistemas distribuidos.

9.2.3. Interfaz del investigador

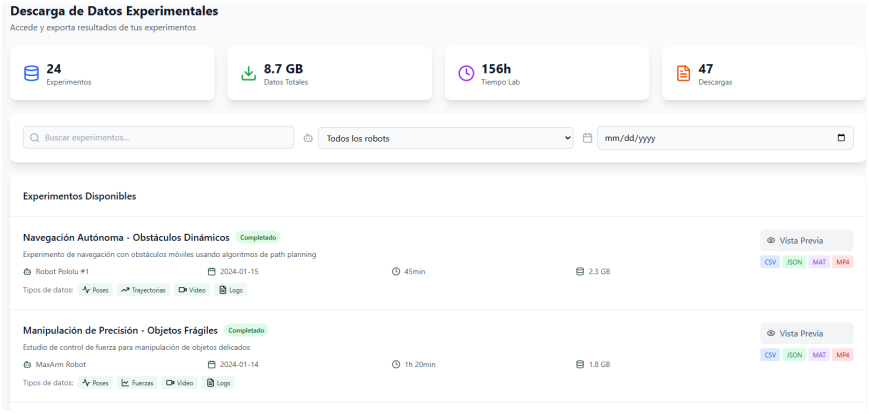
La visualización del investigador está diseñada como un entorno especializado que integra funciones avanzadas de supervisión con herramientas para la gestión y documentación de experimentos. Al acceder por primera vez, el investigador se encuentra

con un panel inicial que resume información clave sobre su actividad en el laboratorio. En este panel se muestran estadísticas acumuladas: número total de experimentos realizados, volumen de datos recolectados (en gigabytes), tiempo efectivo de uso del laboratorio y registro de publicaciones vinculadas a sus trabajos. Estos indicadores no solo informan, sino que también ofrecen una visión clara de la productividad y el uso eficiente de los recursos. Esto permite al investigador evaluar el progreso de sus proyectos y planificar nuevos ensayos.

Otro componente esencial es el módulo de control de cámaras. Al igual que en la interfaz de administrador, los investigadores pueden acceder a transmisiones en tiempo real, ajustar ángulos mediante controles PTZ y grabar secuencias relevantes para su análisis posterior. Esta funcionalidad es especialmente útil en la validación experimental, ya que permite revisar procedimientos, identificar incidencias en la dinámica del laboratorio o documentar evidencia visual para reportes y presentaciones.

La descarga de datos experimentales es una de las funciones más críticas del perfil (Figura 14). Desde esta sección, el investigador puede consultar un listado de experimentos organizado por fecha y responsable, con la opción de previsualizar los datos antes de descargarlos. Los archivos se exportan en distintos formatos —como `.mat`, `.csv` o `.json`—, asegurando compatibilidad con herramientas de análisis estadístico, entornos de programación científica y plataformas de visualización. Además, el sistema registra quién descargó cada experimento y cuándo, lo que refuerza la trazabilidad y promueve la transparencia en la colaboración entre equipos. Así, este módulo va más allá de ser un simple repositorio: se convierte en un sistema integral para la gestión de datos científicos.

Figura 14. Página de descarga de datos experimentales.



Nota. Página destinada a la exportación y descarga de datos experimentales del sistema. Elaboración propia.

También destaca el registro de pruebas, un formulario digital que permite documentar de forma estructurada cada experimento. El investigador ingresa información esencial: nombre del ensayo, robot o cámara utilizada, objetivo principal y observaciones iniciales. Esta estandarización facilita la comparación de resultados a lo largo

del tiempo y, al centralizar los registros en la misma plataforma, contribuye a la construcción de un historial consolidado, útil tanto para fines académicos como para investigaciones a largo plazo.

Finalmente, la visualización incluye un módulo de *logs* MQTT que brinda acceso en tiempo real a la comunicación entre dispositivos y el broker. En conjunto, la visualización del investigador no solo proporciona herramientas de monitoreo y control, sino que constituye un espacio integral para la gestión científica: desde la ejecución de experimentos hasta la sistematización, análisis y trazabilidad de los resultados obtenidos.

CAPÍTULO 10

Acceso a video en tiempo real y control de cámaras del Robotat

La visualización en tiempo real del entorno físico de un laboratorio remoto es esencial para garantizar una interacción sincrónica entre los usuarios y la infraestructura. En el caso del Robotat, el acceso a video en vivo y el control remoto de las cámaras no solo complementan la operación de robots y manipuladores, sino que también permiten a los usuarios verificar directamente las condiciones del entorno. Esta funcionalidad se implementó mediante un sistema híbrido que combina un *backend* robusto desarrollado en `Django` con un microservidor dedicado en `Flask`, responsable del procesamiento y transmisión eficiente del video. Ambos componentes se integran en una interfaz web unificada, adaptada a administradores, estudiantes e investigadores.

10.1. Arquitectura general del sistema

El *backend*, construido con *Django*, actúa como puente entre la interfaz web y las cámaras físicas Amcrest del laboratorio. Con el objetivo de facilitar el mantenimiento y la escalabilidad, se adoptó una arquitectura modular. Por un lado, toda la configuración de las cámaras —incluyendo identificadores, direcciones IP, credenciales y enlaces RTSP— se centralizó en un único componente. Esto permite realizar ajustes, como cambiar una dirección IP o incorporar un nuevo dispositivo, sin afectar al resto del sistema. Por otro lado, la lógica relacionada con la transmisión de video, la validación del estado y el control PTZ se aisló en módulos específicos dedicados a gestionar cada una de estas tareas.

10.2. Transmisión de video en tiempo real

Las cámaras utilizan el protocolo RTSP con la estructura estándar de Amcrest:

```
1        rtsp://admin:UVG12345678@192.168.1.32:554/cam/
   realmonitor?channel=1&subtype=1
2
```

Aquí, `admin` y `UVG12345678` son las credenciales; `192.168.1.32` es la IP; `554` es el puerto RTSP; y `subtype=1` indica que se usa el *substream*: un flujo de menor resolución pero más ligero. Aunque el subtipo 0 ofrece mayor calidad, el subtipo 1 fue elegido por defecto porque permite una transmisión fluida en navegadores web sin saturar la red local.

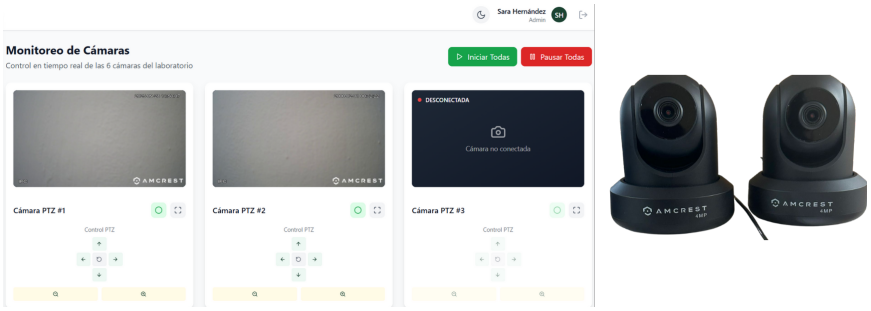
A nivel de arquitectura, la transmisión en vivo está a cargo del microservidor `Flask`, que captura los fotogramas `RTSP`, los decodifica y genera un flujo `MJPEG` eficiente. Este mecanismo permite que el navegador reproduzca el video mediante una etiqueta `HTML <img>`, sin necesidad de *plugins*. Paralelamente, `Django` expone servicios para consultar el estado de las cámaras y enviar comandos PTZ mediante llamadas `CGI` utilizando la biblioteca `requests`.

10.3. Interfaz de usuario y control PTZ

En el *frontend*, la interfaz se adaptó según el rol del usuario. Administradores e investigadores tienen acceso completo: pueden ver el video en vivo —proveniente del microservidor `Flask`— y controlar las cámaras mediante botones direccionales (arriba, abajo, izquierda y derecha). Al hacer clic, el navegador envía un comando `JSON` al *backend* en `Django`, que lo transforma en la instrucción `CGI` correspondiente. La respuesta es inmediata: el movimiento se refleja en tiempo real en el *feed* de video.

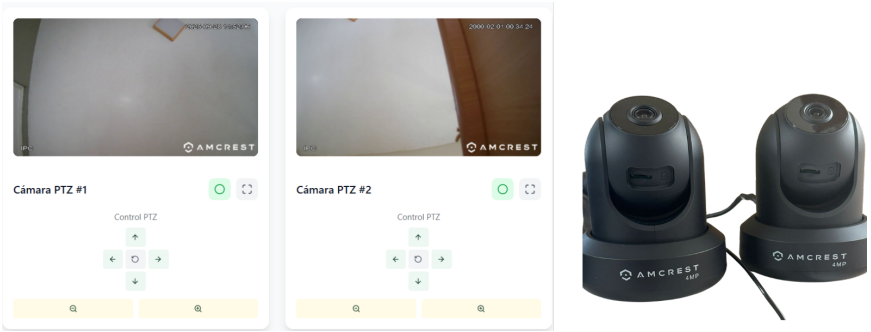
Este comportamiento se ilustra en las Figuras 15 y 16. En la Figura 15 se visualiza la interfaz con dos cámaras orientadas hacia el frente, mostrando su vista estándar. En la Figura 16, las mismas cámaras han sido giradas hacia arriba mediante comandos PTZ y el *feed* de video refleja claramente este cambio de perspectiva. Estas imágenes demuestran la sincronía entre la acción del usuario y la retroalimentación visual.

Figura 15. Visualización en tiempo real de cámaras.



Nota. Interfaz de la plataforma web mostrando el feed en vivo de las cámaras Amcrest junto a los dispositivos físicos empleados en la infraestructura. Elaboración propia.

Figura 16. Control PTZ de cámaras en la plataforma.



Nota. Vista de la página de monitoreo donde se evidencia el control PTZ integrado, permitiendo manipular orientación y zoom de las cámaras Amcrest directamente desde la plataforma. Elaboración propia.

10.4. Diagnóstico y monitoreo de conectividad

El sistema verifica el estado de cada cámara mediante una consulta HTTP al servicio `magicBox.cgi`, usando autenticación *Digest*. En menos de dos segundos, determina si la cámara está en línea y devuelve el resultado en `JSON`, que la interfaz muestra al usuario. Esta funcionalidad está implementada exclusivamente en `Django`, ya que `Flask` se encarga únicamente del *streaming*.

El control PTZ se implementó mediante comandos `HTTP` al servicio `ptz.cgi`. Los comandos incluyen dirección y velocidad y se envían como pulsos breves que inician y detienen el movimiento. Las pruebas confirmaron una latencia mínima: la cámara responde casi al instante, reforzando la sensación de control directo.

En conjunto, estos tres componentes: video en vivo, diagnóstico de conectividad y control PTZ permiten una validación confiable del entorno físico. La plataforma

resultante es inmersiva, estable y preparada para evolucionar.

10.5. Servidor de producción y manejo de concurrencia

Para asegurar estabilidad y rendimiento, cada componente se desplegó con un servidor especializado. `Django`, junto con `Django Channels`, se ejecuta sobre `Daphne`, un servidor `ASGI` capaz de manejar simultáneamente solicitudes `HTTP` y `WebSockets`. Esto permitirá en el futuro monitoreo en tiempo real de eventos del laboratorio. Por su parte, el microservidor `Flask` se ejecuta con `Waitress`, elegido por su compatibilidad con Windows y su capacidad para manejar múltiples conexiones concurrentes sin bloquear el procesamiento de video. Esta separación garantiza una operación continua aún bajo cargas elevadas.

10.6. Resultados y análisis de desempeño

Más allá de la implementación, el análisis de resultados reveló hallazgos clave. Primero, el uso del *substream* (`subtype=1`) redujo significativamente el consumo de ancho de banda sin afectar la utilidad visual, lo que fue crucial para transmitir múltiples cámaras simultáneamente. Segundo, separar el procesamiento de video en `Flask`redujo la carga del *backend*principal y mejoró la estabilidad del sistema, generando un *streaming*continuo incluso con múltiples usuarios conectados.

Durante el análisis, se identificaron oportunidades de mejora. Si bien el rendimiento en red local fue excelente, la transmisión a internet público podría requerir técnicas adicionales de compresión o buffering. Además, el uso de `RTSP` aunque efectivo, limita cierta flexibilidad. En el futuro, protocolos como `WebRTC`o `HLS`podrían facilitar una mejor integración con dispositivos móviles y mejorar la escalabilidad en accesos remotos.

CAPÍTULO 11

Evaluación de alternativas de configuración de red y propuesta de arquitectura remota

La consolidación de un esquema de conexión remota para el laboratorio Robotat requiere una evaluación sistemática de las alternativas de configuración de red disponibles. Se definieron criterios técnicos y de desempeño para comparar distintas arquitecturas, con énfasis en latencia, confiabilidad, seguridad y escalabilidad. Para orientar la selección, se aplicó el MCDA, que permite ponderar múltiples factores y elegir de manera objetiva la arquitectura más adecuada para una conexión remota sincrónica y eficiente. Los resultados obtenidos constituyen la base de la propuesta final de conexión remota presentada a continuación.

11.1. Criterios de evaluación

La selección de la arquitectura de red para el laboratorio Robotat se basa en seis criterios técnicos definidos a partir de sus necesidades operativas: transmisión de video en tiempo real, control sincrónico remoto y acceso seguro. Estos criterios —latencia, confiabilidad, seguridad, escalabilidad, facilidad de implementación y costo— reflejan los requisitos esenciales para garantizar desempeño, estabilidad, protección y sostenibilidad del sistema. Su descripción detallada se presenta en el Cuadro 6, el cual servirá como base para el análisis comparativo mediante el método MCDA.

Cuadro 6. Criterios de evaluación para la arquitectura de red del Robotat.

Criterio	Descripción
Latencia	Tiempo entre el envío de un comando y la respuesta observable del robot. Crítico para la percepción de control en tiempo real en teleoperación y video sincrónico.
Confiabilidad	Capacidad de mantener conexión estable ante fluctuaciones de tráfico o fallos temporales. Esencial para evitar interrupciones en experimentos prolongados.
Seguridad	Protege integridad, confidencialidad y autenticidad mediante cifrado (IPsec/TLS), autenticación institucional y control de accesos. Minimiza la exposición en la red universitaria.
Escalabilidad	Habilidad para integrar nuevos nodos (robots, cámaras, usuarios) sin degradar rendimiento ni requerir reconfiguración total.
Facilidad de implementación	Baja complejidad técnica y uso de infraestructura y herramientas ya disponibles en la institución, especialmente relevante en la fase inicial local.
Costo	Incluye gastos iniciales (equipos, licencias) y operativos (energía, mantenimiento). Determinante para la viabilidad presupuestaria y sostenibilidad a largo plazo.

Nota. Latencia, seguridad y confiabilidad se consideran críticos; los demás, importantes. Elaboración propia.

11.2. Alternativas de configuración de red analizadas

Con base en los criterios definidos, se evaluaron distintas configuraciones. Cada alternativa se analizó teóricamente según su aplicabilidad en el Robotat, seguridad, facilidad de implementación y latencia, adaptando modelos del marco teórico al entorno real del laboratorio.

Alternativa A: NAT con *port forwarding*

El NAT con reenvío de puertos permite conexiones externas redirigiendo tráfico a un servidor local. Es simple y económico, pero expone puertos a Internet, aumentando riesgos de seguridad. Requiere autenticación, cifrado y filtrado para ser seguro. Solo se recomienda en pruebas o entornos controlados dentro del campus [39], [40].

Alternativa B: VPN (IPsec/SSL/TLS)

Las VPN encapsulan el tráfico en túneles cifrados, permitiendo acceso remoto como si el usuario estuviera en la red local. Ofrecen autenticación, confidencialidad e integridad, cumpliendo estándares NIST [43], [44]. Su configuración es más compleja, pero es ideal para teleoperación con video en tiempo real, donde la seguridad es crítica.

Alternativa C: Servidor en la nube

Alojar los servicios del Robotat en la nube (AWS, Azure, etc.) brinda escalabilidad, disponibilidad global y redundancia. Sin embargo, la distancia física incrementa latencia y *jitter*, afectando la sincronía en teleoperación. Además, plantea retos de privacidad y cumplimiento normativo [45], [46].

Alternativa D: Arquitectura híbrida con *edge computing*

Combina un servidor local para control de hardware y uno en la nube para gestión remota. El procesamiento en el *edge* reduce latencia y mejora confiabilidad [47]. Equilibra respuesta local y escalabilidad en la nube, aunque su implementación es más compleja. Es una solución integral para acceso remoto sincrónico y asincrónico.

Alternativa E: Conectividad celular (4G/5G)

Los módulos 4G/5G ofrecen redundancia ante fallas de red fija y permiten movilidad fuera del laboratorio. Las redes 5G logran latencias menores a 10 ms, aptas para control remoto [48]. No obstante, requieren planes de datos robustos y monitoreo constante de desempeño, por lo que se propone como respaldo, no como conexión principal.

11.3. Comparación y análisis preliminar

Con base en los criterios del Cuadro 6 y las alternativas evaluadas, se aplicó el MCDA. Cada alternativa se calificó de 1 a 5 según su desempeño en seis dimensiones clave y los puntajes se ponderaron según la importancia relativa de cada criterio en un entorno de teleoperación sincrónica con video en tiempo real.

La latencia y la seguridad recibieron el mayor peso (0.25 cada una), seguidas por la confiabilidad (0.20). Escalabilidad, facilidad de implementación y costo tuvieron ponderaciones menores (0.15, 0.10 y 0.05, respectivamente).

Cuadro 7. Análisis multicriterio (MCDA) de alternativas de red para Robotat.

Criterio	**Peso**	**NAT**	**VPN**	**Nube**	**Híbrida**	**4G/5G**
Latencia	0.25	4	4	3	**5**	3
Seguridad	0.25	2	**5**	4	**5**	3
Confiabilidad	0.20	3	4	5	**5**	3
Escalabilidad	0.15	2	3	5	4	4
Facilidad de implementación	0.10	**5**	3	4	3	4
Costo de implementación	0.05	**5**	3	3	3	2
Puntaje ponderado total	**1.00**	3.35	4.15	4.15	**4.65**	3.25

Nota. La arquitectura híbrida obtiene el mejor equilibrio entre latencia, seguridad y confiabilidad. Elaboración propia.

La arquitectura híbrida con *edge computing* resultó la opción más adecuada (4.65), al integrar procesamiento local para tareas sensibles a latencia y recursos remotos para gestión y escalabilidad. Las alternativas VPN y nube empataron (4.15): la primera prioriza seguridad institucional; la segunda, flexibilidad. En cambio, NAT y 4G/5G presentan limitaciones en seguridad o estabilidad, lo que las descarta para operación sincrónica crítica.

El análisis respalda un enfoque híbrido con túnel seguro (VPN) y procesamiento en el *edge* como estrategia óptima para equilibrar rendimiento, protección y sostenibilidad.

11.4. Propuesta de arquitectura orientada a la conexión remota

Se propone una arquitectura híbrida que combina un servidor local en el laboratorio Robotat con una VPN institucional y un proxy inverso (NGINX). Esta solución garantiza baja latencia, alta seguridad y escalabilidad, alineada con los requisitos de teleoperación y transmisión de video en tiempo real.

Descripción general

La propuesta centraliza toda la lógica crítica en un entorno local que integra tres componentes principales: un servidor ejecutando el *broker* `MQTT` un *backend* en `Django`para autenticación y gestión de usuarios, y un microservidor `Flask` encargado del procesamiento y transmisión `MJPEG` desde las cámaras Amcrest. Esta organización reduce dependencias externas, optimiza el rendimiento de servicios que requieren respuesta inmediata y preserva la estabilidad del laboratorio incluso bajo cargas elevadas.

El acceso remoto se canaliza exclusivamente a través de la VPN institucional, configurada con protocolos IPsec, SSL o TLS. Esta capa garantiza que únicamente usuarios con credenciales institucionales (`@uvg.edu.gt`) puedan conectarse a la red interna, interactuando con el sistema como si estuvieran físicamente presentes en el laboratorio. La VPN actúa como el primer filtro de seguridad, evitando la exposición directa de los servicios del Robotat a Internet.

Para reforzar aún más la protección, todas las solicitudes entrantes se gestionan mediante un servidor `NGINX` configurado como proxy inverso. `NGINX` aplica certificados `SSL/TLS`, enruta el tráfico hacia los servicios correspondientes y bloquea accesos directos a puertos internos. De esta manera, los componentes esenciales —`Django`, `MQTT` y `Flask`— permanecen aislados tras una capa de control que expone únicamente las rutas necesarias para la operación remota.

Complementando esta arquitectura, una base de datos local almacena telemetría, sesiones, registros de interacción y eventos relevantes para auditoría. Esta decisión elimina la dependencia de servicios externos, asegura trazabilidad completa y facilita el cumplimiento de políticas institucionales sobre manejo de información. Además, el repositorio centralizado permite analizar patrones de uso y realizar validaciones posteriores del comportamiento del sistema.

El flujo de conexión comienza cuando el usuario se autentica en la `VPN` e ingresa al portal web. `NGINX` recibe la solicitud y la redirige al *backend*, donde `Django` verifica las credenciales y habilita el acceso a los servicios autorizados según el rol del usuario. La comunicación con robots y sensores se gestiona mediante `MQTT`, mientras que el video en vivo se entrega desde el microservidor `Flask` en formato `MJPEG`. Cada interacción genera un registro en la base de datos, garantizando transparencia y control operativo.

Esta arquitectura ofrece ventajas clave: el procesamiento crítico se mantiene dentro de la red local, asegurando baja latencia; la combinación de VPN y cifrado SSL/TLS proporciona seguridad reforzada; y el diseño modular permite incorporar nuevos robots, cámaras u otros dispositivos sin reestructuraciones profundas. Asimismo, la solución es compatible con la infraestructura existente del Robotat y está preparada para evolucionar hacia modelos híbridos con componentes en la nube en fases futuras.

Entre las líneas de trabajo futuras se plantea la migración del sistema a un servidor físico con Ubuntu y NGINX, la configuración de la VPN institucional con módulos FIPS 140-3, la evaluación de nodos *edge* para extender la infraestructura hacia la nube, la medición de latencia bajo carga con usuarios remotos y la automatización del despliegue mediante contenedores Docker. En conjunto, esta propuesta sienta las bases para un acceso remoto sincrónico, seguro y escalable, ampliando el alcance del laboratorio Robotat más allá de sus límites físicos.

CAPÍTULO 12

Conclusiones

- La consolidación de los servicios en un servidor dedicado permitió centralizar la comunicación mediante un *broker* `MQTT`, lo que se verificó mediante la publicación y suscripción exitosa de mensajes entre múltiples clientes en la red local.
- El uso de `MQTT` como núcleo de la infraestructura mejoró la organización de los flujos de datos: la implementación de tópicos diferenciados permitió segmentar de forma clara la telemetría y el control de robots.
- El formato estandarizado de los paquetes `MQTT` aseguró consistencia en los datos, demostrada por la correcta transmisión de metadatos (origen, comando, identificador) en distintos escenarios.
- La implementación de `Mosquitto` como *broker* `MQTT` en el servidor dedicado demostró un rendimiento sólido y consistente, manteniendo un desempeño estable incluso con múltiples clientes publicando y suscribiéndose simultáneamente.
- Se implementó el acceso al video en tiempo real y el control de cámaras físicas. El sistema demostró ser funcional y suficiente para monitorear el entorno del laboratorio, validándose la transmisión de datos mediante visualización continua, grabación del flujo y respuesta inmediata a comandos PTZ.
- Al configurar el flujo `RTSP` con `subtype=1`, se redujo el consumo de ancho de banda y se logró una transmisión más fluida que con el flujo predeterminado (`subtype=0`). La mejora se validó mediante la reducción de latencia y la mayor estabilidad observadas durante las pruebas de visualización continua.
- Durante las pruebas de transmisión de video, se identificó un leve retardo asociado al procesamiento y envío del flujo RTSP. Si bien no afectó la supervisión ni el control básico de las cámaras, este hallazgo resalta la importancia de optimizar la infraestructura para futuras implementaciones remotas, donde la latencia cobrará un papel más crítico.

- Se implementó autenticación con *tokens* `JWT` (acceso y actualización) y una lista negra para revocar accesos al cerrar sesión o cambiar credenciales. El sistema demostró su eficacia al bloquear accesos con *tokens* revocados y forzar cierres de sesión en múltiples clientes.
- La implementación de una lista negra permitió revocar *tokens* de forma inmediata del lado del servidor, mitigando riesgos por robo o filtración. Su eficacia se demostró al impedir accesos tras el cierre de sesión o el cambio de contraseña.
- La implementación del *frontend* con `Vite`, `React` y `Tailwind CSS` redujo los tiempos de carga mediante un *bundling* más eficiente y permitió mantener coherencia visual con componentes reutilizables y estilos consistentes.
- El *backend* en `Django` centralizó la autenticación (mediante emisión, verificación y rotación de *tokens*) y la autorización (por roles y permisos en cada vista o *endpoint*), demostrando un comportamiento correcto en pruebas de acceso permitido y denegado.
- El control del Pololu 3Pi+ mediante `MQTT` funcionó correctamente, validando la comunicación en tiempo real entre la interfaz web, el *broker* y el robot y demostrando que la infraestructura es adecuada para operaciones de control de robots móviles.
- Los resultados obtenidos demostraron que la arquitectura remota propuesta es técnicamente viable, ya que todos los servicios núcleo (`MQTT`, `Django`, `Flask` y `React`) operaron correctamente en el servidor local y son completamente compatibles con una capa adicional de acceso remoto mediante `VPN` y proxy inverso.
- El análisis realizado demuestra que la arquitectura remota puede operar con baja latencia si se combina con optimizaciones de video como WebRTC o HLS según el tipo de interacción. Esto sienta las bases para experiencias de teleoperación fluidas y confiables.

CAPÍTULO 13

Recomendaciones

- **Ampliar la infraestructura hacia un entorno remoto real:** dado que el trabajo se limitó a la consolidación local, se recomienda extender la arquitectura hacia esquemas de acceso remoto. Esto incluye la implementación de túneles `VPN`, balanceo de carga y despliegue de servicios en la nube, lo que permitirá validar el rendimiento fuera de la red interna del laboratorio.
- **Optimizar la transmisión de video en tiempo real:** el sistema cumplió el objetivo de habilitar el acceso a cámaras Amcrest, pero para la evolución remota se aconseja evaluar protocolos más eficientes (como `WebRTC` o `HLS`) y considerar mecanismos de compresión adaptativa para mantener baja latencia y estabilidad de *streaming* en condiciones de red variables.
- **Explorar nuevas líneas de investigación en interfaces de usuario:** dado que la plataforma web ya incluye autenticación y vistas diferenciadas, sería valioso incorporar accesibilidad, soporte multilenguaje y experimentación con interfaces inmersivas (realidad aumentada o entornos 3D) para enriquecer la experiencia de usuarios remotos.
- **Fortalecer la seguridad de la autenticación:** partiendo de la implementación existente de listas negras y expiración de *tokens* `JWT`, se recomienda ampliar el esquema hacia una rotación periódica de claves privadas/públicas (*key rotation*) y la incorporación de un sistema de monitoreo en tiempo real para detectar patrones de ataque (p. ej., múltiples intentos fallidos desde una misma IP o uso indebido de *tokens* comprometidos). Asimismo, podría explorarse la autenticación multifactor (MFA) como capa adicional de seguridad en escenarios de conexión remota.
- **Uso de ambientes virtuales:**

  Se recomienda desarrollar y ejecutar los servicios (`Django`, `Flask` y *scripts* `MQTT`) dentro de entornos virtuales de Python para mantener controladas las dependencias, evitar conflictos entre versiones y facilitar el despliegue en otros servidores.

- **Uso de ambientes virtuales:** Se recomienda automatizar procesos comunes (iniciar el *broker*, activar el entorno virtual, levantar `Django`, iniciar `Flask`) mediante scripts de *shell* o *lotes*, facilitando la puesta en marcha del sistema por parte de nuevos usuarios.

CAPÍTULO 14

Referencias

[1] Y. Zhao y Z. Ye, «Effectiveness of virtual laboratory in engineering education: A meta-analysis,» *Journal of Educational Technology Development and Exchange*, vol. 15, n.º 1, págs. 1-15, 2022. DOI: 10.1007/s10956-022-09976-8

[2] P. Pickem Daniel, M. Mote, Y. Qi, T. Wu, M. Wang y M. Egerstedt, «The Robotarium: A remotely accessible swarm robotics research testbed,» en *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, págs. 1699-1706. DOI: 10.1109/ICRA.2017.7989200

[3] J. García-Zubía, P. Orduña, D. López de Ipiña, U. Hernández e I. Trueba, «Evolution of the WebLab at the University of Deusto,» en *EWME 2006*, Estocolmo, Suecia, jun. de 2006.

[4] J. B. Da Silva, S. M. S. Bilessimo y U. F. De Santa Catarina, *Integração da tecnologia e cultura Maker: proposta de reconfiguração de espaço físico do laboratório de experimentação remota - RExLab*, https://repositorio.ufsc.br/handle/123456789/215116, 2019.

[5] CWI Construction. «Georgia Tech Robotarium. »dirección: https://www.cwiconstruction.com/projects/georgia-tech-robotarium

[6] M. Egerstedt, C. Granas, A. Hamilton, D. Wilson y D. Pickem, «The Robotarium: A Remotely-Accessible, Multi-Robot Testbed for Control Research and Education,» *IEEE Transactions on Robotics*, vol. 38, n.º 6, págs. 3801-3817, 2022. DOI: 10.1109/TRO.2022.3198566

[7] J. García-Zubía, D. Ponta, U. Hernández, P. Orduña e I. Angulo, *WebLab-GPIB en la Universidad de Deusto*, Universidad de Deusto. [Documento PDF], 2008.

[8] I. Field, *Brazilian university develops a remote experimentation tool for public education - In The Field*, https://www.inthefieldstories.net/brazilian-university-develops-a-remote-experimentation-tool-to-support-public-education, 2016.

[9] J. B. Da Silva, I. N. Da Silva, S. M. S. Bilessimo y J. B. Da Mota Alves, «Academic Path Linked to Research and Extension: The Experience of the Remote Experimentation Laboratory (Rexlab) of The Federal University of Santa Catarina,» *Revista de Gestão Social E Ambiental*, vol. 18, n.º 9, e06369, 2024. DOI: 10.24857/rgsa.v18n9-063

[10] J. E. Pu Aguilera, *Desarrollo de herramientas de programación y simulación para los agentes robóticos Pololu 3Pi+ dentro del ecosistema Robotat*, Trabajo de graduación graduación de licenciatura, 2023.

[11] P. Barrera. «16 datos sobre el Robotat. »dirección: https://noticias.uvg.edu.gt/datos-robotat-habitat-robotica-cit-116/

[12] S. Lakshminarayana, «Securing the IoT application layer from an MQTT protocol perspective,» *IEEE Communications Surveys and Tutorials*, 2024. DOI: 10.1109/COMST.2024.3372630 dirección: https://doi.org/10.1109/COMST.2024.3372630

[13] M. Saide, *Exploring the TCP/IP protocol suite: Architecture, dominance, and future challenges in data communication*, https://doi.org/10.2139/ssrn.4885418, 2024.

[14] A. F. Gentile, D. Macrì, D. L. Carnì, E. Greco y F. Lamonaca, «A network performance analysis of MQTT security protocols with constrained hardware in the dark net for DMS,» *Applied Sciences*, vol. 14, n.º 18, pág. 8501, 2024. DOI: 10.3390/app14188501 dirección: https://doi.org/10.3390/app14188501

[15] A. E. Corona. «Estándares de video: competencia de marcas y tecnología.» Fecha de aceptación: 2 de septiembre de 2004, Universidad Nacional Autónoma de México. dirección: https://www.revista.unam.mx/vol.5/num8/art51/sep_art51.pdf

[16] Internet Engineering Task Force, *RFC 9293 – Transmission Control Protocol (TCP)*, https://www.rfc-editor.org/info/rfc9293, 2022.

[17] W. R. Stevens y K. R. Fall, *TCP/IP Illustrated, Volume 1: The Protocols*, 2.ª ed. Boston, MA: Addison-Wesley, 2011.

[18] J. F. Kurose y K. W. Ross, *Computer Networking: A Top-Down Approach*, 8.ª ed. Boston, MA: Pearson, 2021.

[19] D. E. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architecture*, 7.ª ed. Boston, MA: Pearson, 2018.

[20] Amazon Web Service, *MQTT - AWS IoT Core*, https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html, 2022.

[21] A. Bassi, *Introducción a MQTT*, https://www.gotoiot.com/pages/articles/mqtt_intro/content.html, feb. de 2021.

[22] R. A. Light, «Mosquitto: server and client implementation of the MQTT protocol,» *Journal of Open Source Software*, vol. 2, n.º 13, pág. 265, 2017. DOI: 10.21105/joss.00265

[23] L. Simmons. «Front-end vs. back-end development: What's the difference? »Dirección: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/

[24] D. M. Ng, «Smart Application Using MQTT Protocol for Industrial IoT and Retail,» *Journal of Science & Technology*, vol. 5, n.º 1, 2024.

[25] I. Sahmi, A. Abdellaoui, T. Mazri y N. Hmina, «MQTT-PRESENT: Approach to secure internet of things applications using MQTT protocol,» *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, n.º 5, págs. 4577-4586, 2021.

[26] E. Buetas Sanjuan, I. Abad Cardiel, J. A. Cerrada y C. Cerrada, «Message Queuing Telemetry Transport (MQTT) Security: A Cryptographic Smart Card Approach,» *IEEE Access*, vol. 8, págs. 115 051-115 062, 2020.

[27] Eclipse Foundation, *Eclipse Mosquitto: An open source MQTT broker*, `https://mosquitto.org/`, 2025.

[28] U. Hunkeler, H. L. Truong y A. Stanford-Clark, «Efficient MQTT broker design for IoT devices with resource constraints,» *IEEE Internet of Things Journal*, vol. 10, n.º 4, págs. 3300-3312, 2023. DOI: `10.1109/JIOT.2023.3257790` dirección: `https://doi.org/10.1109/JIOT.2023.3257790`

[29] J. Smith y M. Lee, «Securing MQTT communications with TLS/SSL: Best practices,» *Journal of Network Security*, vol. 18, n.º 1, págs. 45-60, 2024. DOI: `10.1016/j.jnsec.2024.01.005` dirección: `https://doi.org/10.1016/j.jnsec.2024.01.005`

[30] R. Johnson, «MQTT persistence and session management: Ensuring reliability in IoT networks,» *International Journal of Distributed Sensor Networks*, vol. 20, n.º 2, págs. 112-125, 2024. DOI: `10.1177/15501477231234567` dirección: `https://doi.org/10.1177/15501477231234567`

[31] Emqx Corporation. «MQTT broker comparison: Mosquitto vs enterprise solutions. »dirección: `https://www.emqx.com/en/blog/emqx-vs-mosquitto-2023-mqtt-broker-comparison`

[32] Amcrest Technologies, *User Manual (IP4M-1041)*, `https://support.amcrest.com/hc/en-us/articles/360037230732-User-Manual-IP4M-1041`, 2019.

[33] Amcrest Technologies, *Amcrest HTTP API SDK*, `https://support.amcrest.com/hc/en-us/articles/4410620161933-Amcrest-HTTP-API-SDK`, 2021.

[34] Eyeson. «What is RTSP (Real Time Streaming Protocol)? »Dirección: `https://blog.eyeson.com/what-is-rtsp-real-time-streaming-protocol`

[35] Reolink. «RTSP: Real-Time Streaming Protocol Overview. »dirección: `https://reolink.com/blog/what-is-rtsp/`

[36] Y. W. Syaifudin, I. F. Rozi, R. Ariyanto, E. Rohadi y S. Adhisuwignjo, «Study of performance of real time streaming protocol (RTSP) in learning systems,» *International Journal of Engineering & Technology*, vol. 7, n.º 2.25, págs. 74-77, 2018. DOI: `10.14419/ijet.v7i2.25.11992` dirección: `https://doi.org/10.14419/ijet.v7i2.25.11992`

[37] GllmAR, *Amcrest RTSP URL*, abr. de 2024. dirección: `https://gist.github.com/gllmAR/9fe45c83c72a4d1a226d1270009ae43f`

[38] TutorHunt. «Real-time Streaming Protocol (RTSP) performance & security in video. »dirección: `https://www.tutorhunt.com/resource/26950/`

[39] H. Wang, Y. Xue, X. Feng, C. Zhou y X. Mi, *Port Forwarding Services Are Forwarding Security Risks*, 2024. arXiv: `2403.16060 [cs.CR]`. dirección: `https://arxiv.org/abs/2403.16060`

[40] P. Srisuresh y K. Egevang, *Traditional IP Network Address Translator (Traditional NAT)*, https://www.rfc-editor.org/rfc/rfc3022.html, 2001.

[41] A. A. Kist, A. Maiti, A. Maxwell, L. Orwin y V. Potkonjak, «Overlay network architectures for peer-to-peer remote access laboratories,» en *2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2015, págs. 221-226. DOI: 10.1109/REV.2015.7087281 dirección: https://doi.org/10.1109/REV.2015.7087281

[42] K. Scarfone y P. Hoffman, «Guidelines on Firewalls and Firewall Policy,» National Institute of Standards y Technology, inf. téc. NIST SP 800-41 Rev. 1, 2009. dirección: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-41r1.pdf

[43] S. Frankel, P. Hoffman, A. Orebaugh y R. Park, «Guide to SSL VPNs,» National Institute of Standards y Technology, inf. téc. NIST SP 800-113, 2008. DOI: 10.6028/NIST.SP.800-113 dirección: https://doi.org/10.6028/NIST.SP.800-113

[44] E. Barker, Q. Dang, S. Frankel, K. Scarfone y P. Wouters, «Guide to IPsec VPNs,» National Institute of Standards y Technology, inf. téc. NIST SP 800-77 Rev. 1, 2020. DOI: 10.6028/NIST.SP.800-77r1 dirección: https://doi.org/10.6028/NIST.SP.800-77r1

[45] C. Rejón, S. Martín y A. Robles-Gómez, «Easy development of Industry 4.0 remote labs,» *Electronics*, vol. 13, n.º 8, pág. 1508, 2024. DOI: 10.3390/electronics13081508 dirección: https://doi.org/10.3390/electronics13081508

[46] P. Mell y T. Grance, «The NIST Definition of Cloud Computing,» National Institute of Standards y Technology, inf. téc. NIST SP 800-145, 2011. dirección: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[47] D. Sabella, A. Li, H. Lee, L. Cominardi, Q. Huang y E. Pateromichelakis, «MEC Support towards Edge-Native Design and Operation,» ETSI - European Telecommunications Standards Institute, inf. téc. White Paper No. 55, jun. de 2023. dirección: https://www.etsi.org/images/files/ETSIWhitePapers/ETSI-WP55-MEC_support_towards_Edge_native.pdf

[48] B. Kizilkaya, G. Zhao, Y. A. Sambo, L. Li y M. A. Imran, «5G-enabled Education 4.0: Enabling technologies, challenges, and solutions,» *IEEE Access*, vol. 9, págs. 17826-17840, 2021. DOI: 10.1109/ACCESS.2021.3053172 dirección: https://doi.org/10.1109/ACCESS.2021.3053172

[49] J. E. Cardona y J. G. Hoyos, «A remote laboratory platform for teaching automation and control,» en *IEEE 40th Central America and Panama Convention (CONCAPAN XL)*, 2022, págs. 1-6. DOI: 10.1109/CONCAPAN48024.2022.9997707 dirección: https://doi.org/10.1109/CONCAPAN48024.2022.9997707

[50] I.-E. Wu, «Authentication in Web Applications,» Bachelor's thesis, Metropolia University of Applied Sciences, 2023. dirección: https://www.theseus.fi/bitstream/handle/10024/813453/Wu_I-En.pdf

[51] Authgear. «JWT Authentication: A secure & scalable solution for modern applications. »dirección: https://www.authgear.com/post/jwt-authentication-a-secure-scalable-solution-for-modern-applications

[52] A. Sharma, D. V. Shrivastava, A. Pandey y A. Sharma, «Providing authentication using JSON web tokens for enhancing user security,» *International Journal of Research Publication and Reviews*, vol. 5, n.º 4, págs. 5309-5312, 2024. dirección: https://ijrpr.com

[53] M. Jones, J. Bradley y N. Sakimura, *RFC 7519 - JSON Web Token (JWT)*, https://datatracker.ietf.org/doc/html/rfc7519, 2015.

[54] JWT.io. «JSON Web Token Introduction. »dirección: https://jwt.io/introduction

[55] J. Díaz, M. López y P. Ramírez, *Evaluating the security of RFC 8725: An analysis of JWT Best Practices*, 2025. dirección: https://www.diva-portal.org/smash/get/diva2:1951665/FULLTEXT01.pdf

[56] Auth0. «JSON Web Tokens. »dirección: https://auth0.com/learn/json-web-tokens

[57] G. Eduarda. «JWT in Practice – Part 2: Refresh Tokens, Expiration, and Best Practices. »dirección: https://dev.to/gabrielle_eduarda_776996b/jwt-in-practice-part-2-refresh-tokens-expiration-and-best-practices-20p2

[58] P. Singh, *Assessment And Execution Manual For Json Web Token (Jwt) Verification*, 2024. dirección: https://ilkogretim-online.org/index.php/pub/article/download/7546/7251/14383

[59] Redis. «JSON Web Tokens (JWT) are Dangerous for User Sessions. »dirección: https://redis.io/blog/json-web-tokens-jwt-are-dangerous-for-user-sessions/

[60] Curity. «JWT Security Best Practices. »dirección: https://curity.io/resources/learn/jwt-best-practices/

[61] V. Abhinaya, K. Akhila, K. Deepika y S. Hamsa, *Comparative analysis of full-stack Python frameworks: Django, Flask, and FastAPI*, 2025.

[62] M. Idris, *A generic review of web technology: Django and Flask*, 2023.

[63] Kerry Doyle. «Django vs Flask: Comparing Python web frameworks. »dirección: https://www.techtarget.com/searchapparchitecture/tip/Django-vs-Flask-Comparing-Python-web-frameworks

[64] Simplilearn. «Django vs. Flask: Understanding the major differences. »dirección: https://www.simplilearn.com/flask-vs-django-article

[65] P. Chapkovski y E. Kujansuu, «Real-time interactions in oTree using Django Channels,» *Journal of Behavioral and Experimental Finance*, vol. 21, pág. 100 260, 2019. dirección: https://www.sciencedirect.com/science/article/abs/pii/S2214635018302612

[66] Pylons Project, *Waitress 3.0.2 documentation*, 2025. dirección: https://docs.pylonsproject.org/projects/waitress/en/latest/

[67] A. Mardani, A. Jusoh, K. M. Nor, Z. Khalifah, N. Zakwan y A. Valipour, «Multiple criteria decision-making techniques and their applications–a review of the literature from 2000 to 2014,» *Economic Research-Ekonomska Istraživanja*, vol. 28, n.º 1, págs. 516-571, 2015. DOI: 10.1080/1331677X.2015.1075139

[68] R. Baltussen, M. P. Jansen y Bijlmakers, «Value Assessment Frameworks for HTA Agencies: The Organization of Evidence-Informed Deliberative Processes,» *Value in Health*, vol. 22, n.º 4, págs. 471-478, 2019. DOI: 10.1016/j.jval.2018.11.003

[69] S. Chaube, S. Pant, A. Kumar, S. Uniyal, M. K. Singh y K. Kotecha, «An Overview of Multi-Criteria Decision Analysis and the Applications of AHP and TOPSIS Methods,» *International Journal of Mathematical, Engineering and Management Sciences*, vol. 9, n.º 3, págs. 581-615, 2024. DOI: 10.33889/IJMEMS.2024.9.3.030

[70] National Institute of Standards and Technology. «blacklist. »dirección: https://csrc.nist.gov/glossary/term/blacklist

[71] K. Moriarty, B. Kaliski y A. Rusch, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, https://www.rfc-editor.org/rfc/rfc8018.html, ene. de 2017.

[72] K. Scarfone y M. Souppaya, «Cybersecurity Log Management Planning Guide,» National Institute of Standards y Technology, inf. téc. SP 800-92 Rev. 1, oct. de 2023. dirección: https://csrc.nist.gov/pubs/sp/800/92/r1/ipd

[73] D. S. Gardón. «The Complete JavaScript Module Bundlers Guide. »dirección: https://snipcart.com/blog/javascript-module-bundler

CAPÍTULO 15

Anexos

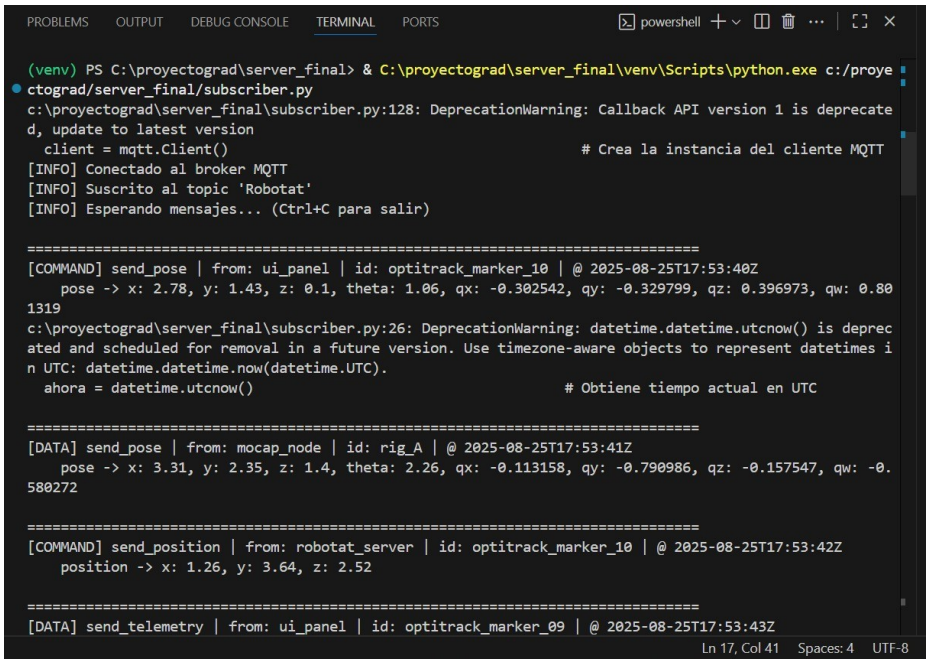
En este apartado se incluyen capturas de la ejecución de scripts en Python utilizados para la comunicación mediante MQTT —que muestran tanto la publicación como la recepción de mensajes a través del *broker*—, así como ejemplos de la interfaz de usuario en sus distintas modalidades: vistas de administrador, investigador y estudiante.

Este material tiene como finalidad ofrecer evidencia práctica de la implementación descrita en el cuerpo principal del documento, facilitando una comprensión más clara de aspectos como la interacción en consola, la operación de la plataforma web, la transmisión en tiempo real, el control de cámaras y la gestión de recursos. Los anexos, por lo tanto, sirven como soporte visual y técnico a los hallazgos y funcionalidades presentados anteriormente.

Figura 17. Vista del módulo de análisis y monitoreo de interfaz web.

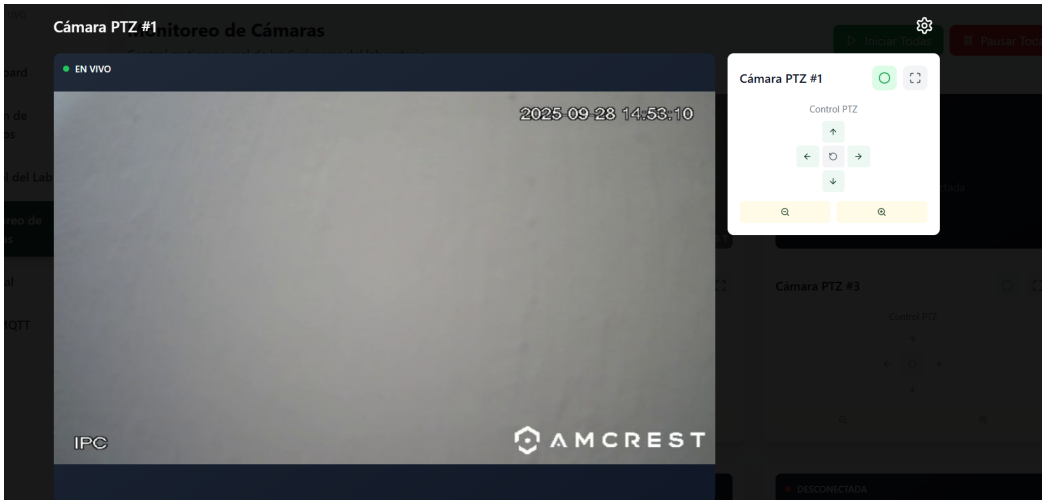
Nota. La interfaz permite supervisar marcadores activos y evaluar el comportamiento del flujo de datos mediante indicadores visuales de calidad. Incluye estadísticas generales, integridad de transmisión y detección de duplicados o pérdidas. Elaboración propia.

Figura 18. Terminal del *subscriber* mostrando recepción de mensajes desde el *broker* MQTT.



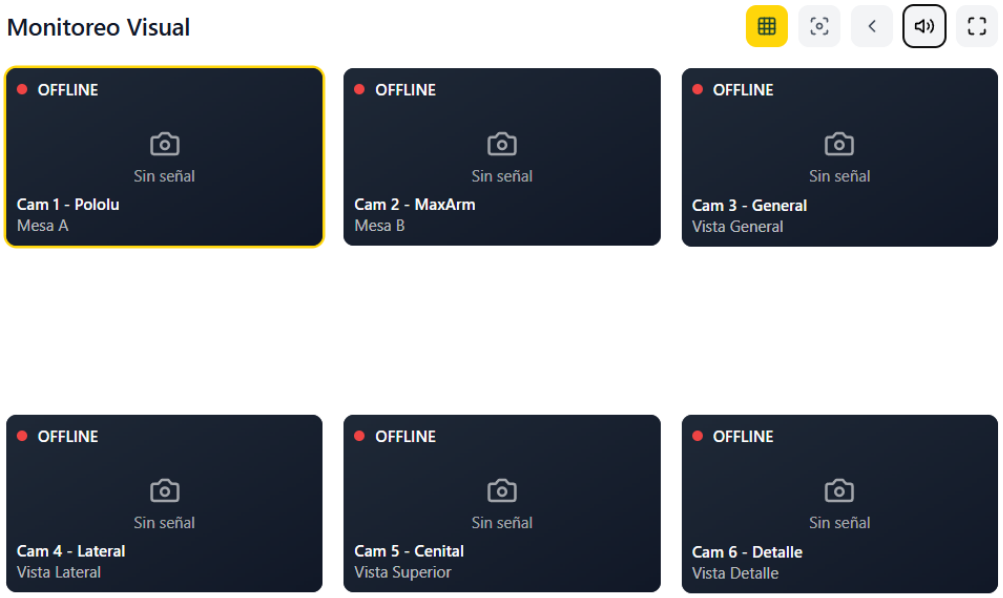
Nota. Ejecución en Python del subscriber mostrando la suscripción a tópicos y la recepción de mensajes. Elaboración propia.

Figura 19. Video en tiempo real versión agrandada de la vista de administrador.



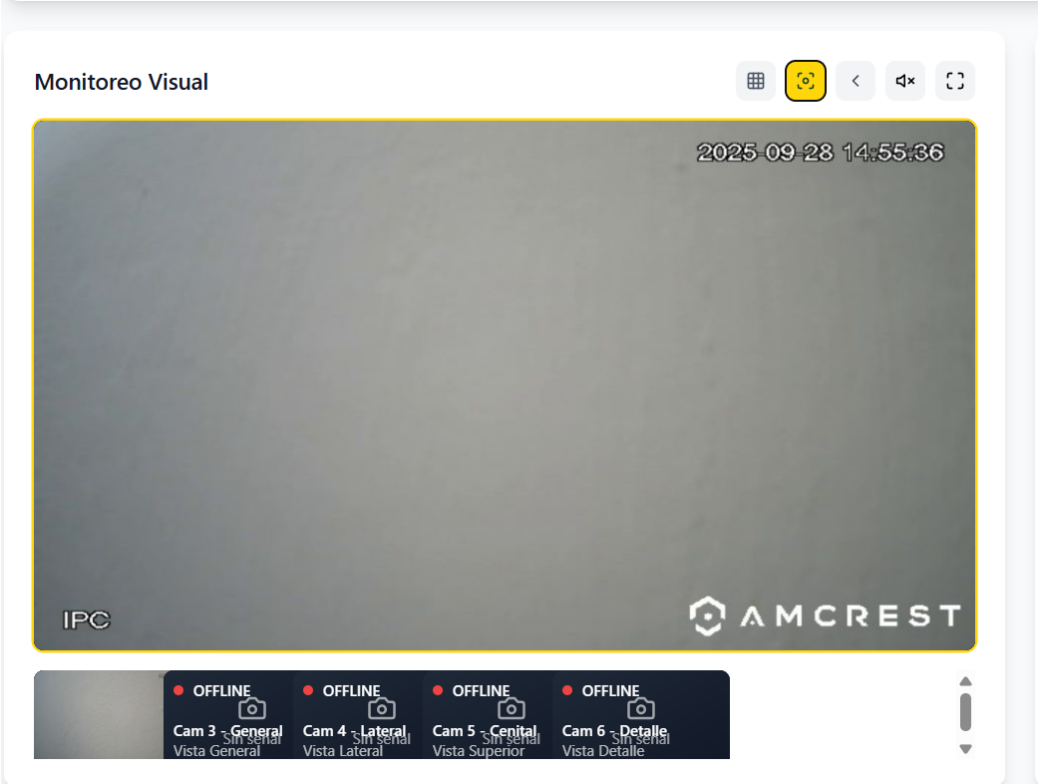
Nota. En este Anexo se muestra el video en tiempo real de una de las cámaras en la vista de administrador.Elaboración propia.

Figura 20. Visualización cuadrícula vista de estudiante.



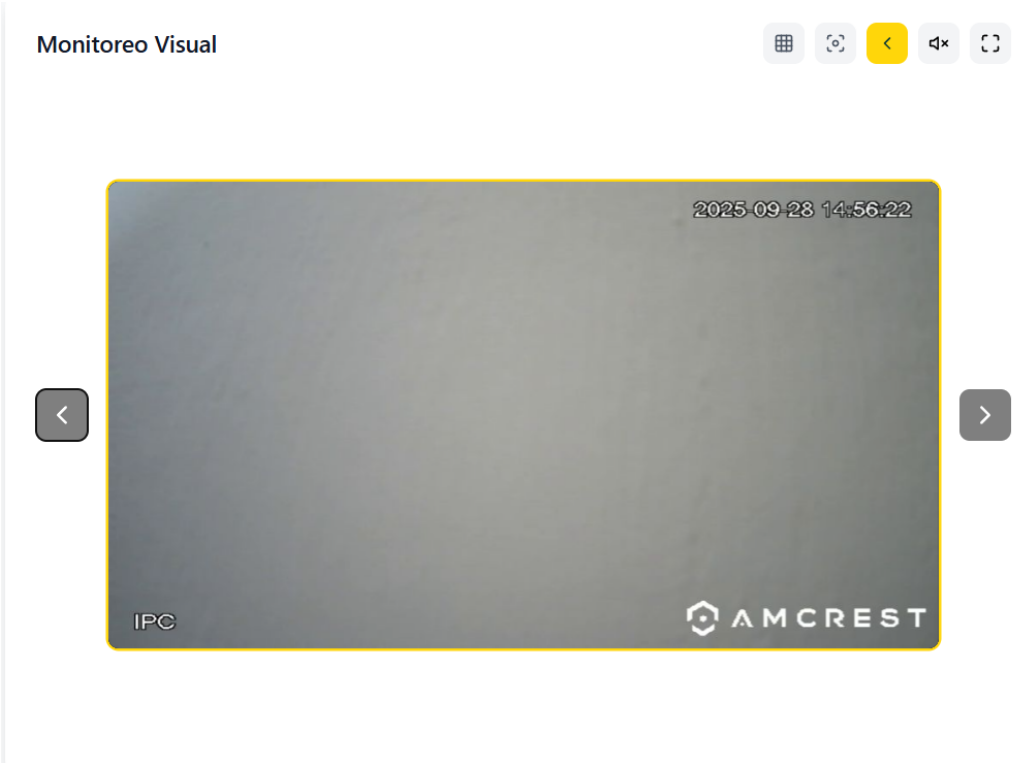
Nota. Visualización modo cuadrícula para vista de estudiante. Elaboración propia.

Figura 21. Modo de monitoreo visual: cámara principal ampliada y vistas en miniatura de las restantes.



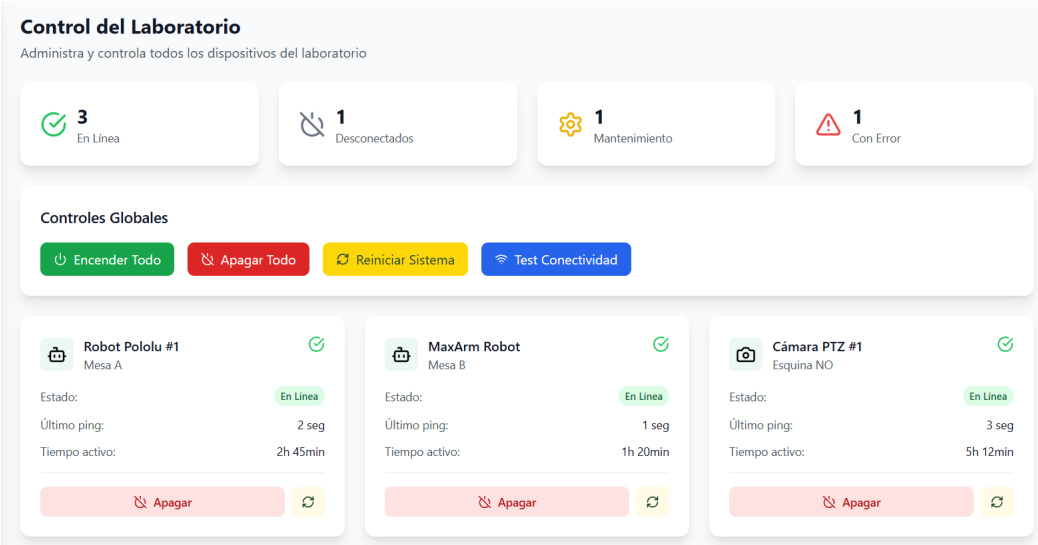
Nota. Ejemplo de la visualización en la plataforma mostrando una cámara ampliada y las demás como miniaturas. Elaboración propia.

Figura 22. Visualización de cámaras en carrusel: una cámara a la vez con navegación lateral.



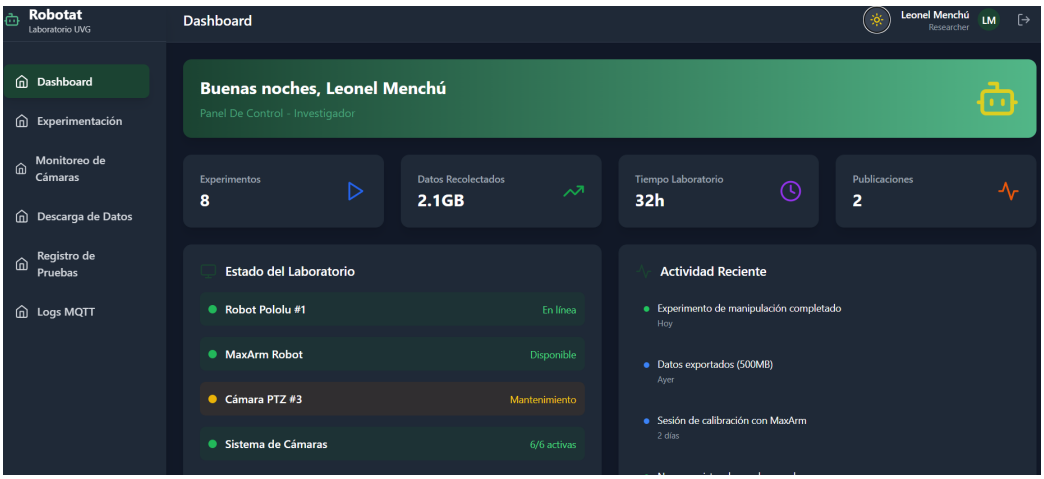
Nota. Ejecución en la plataforma en modalidad carrusel, permitiendo alternar entre cámaras de forma secuencial. Elaboración propia.

Figura 23. Interfaz de control del laboratorio disponible para administradores.



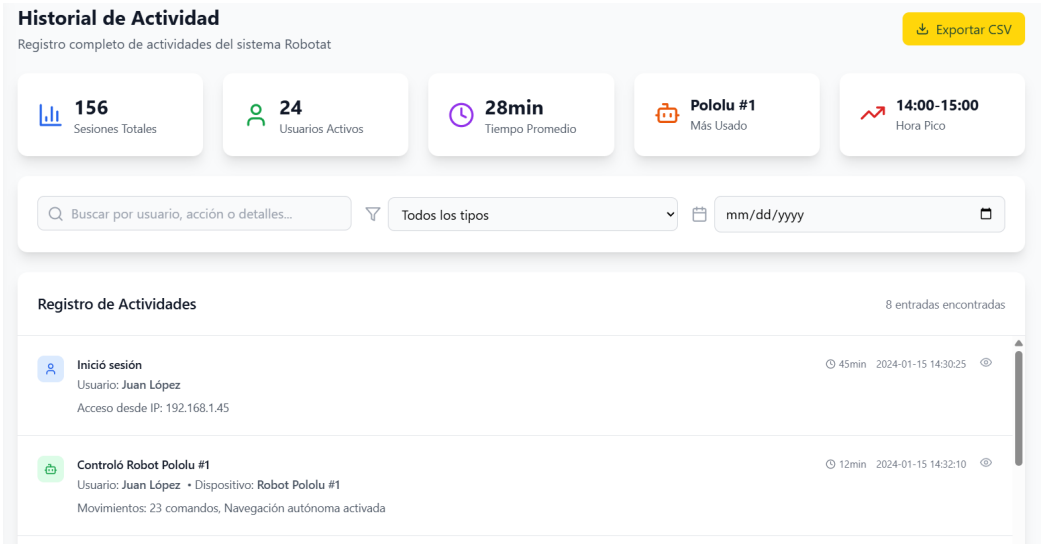
Nota. Vista del panel administrativo con control global de la infraestructura del laboratorio. Elaboración propia.

Figura 24. Panel de control del investigador en modo nocturno



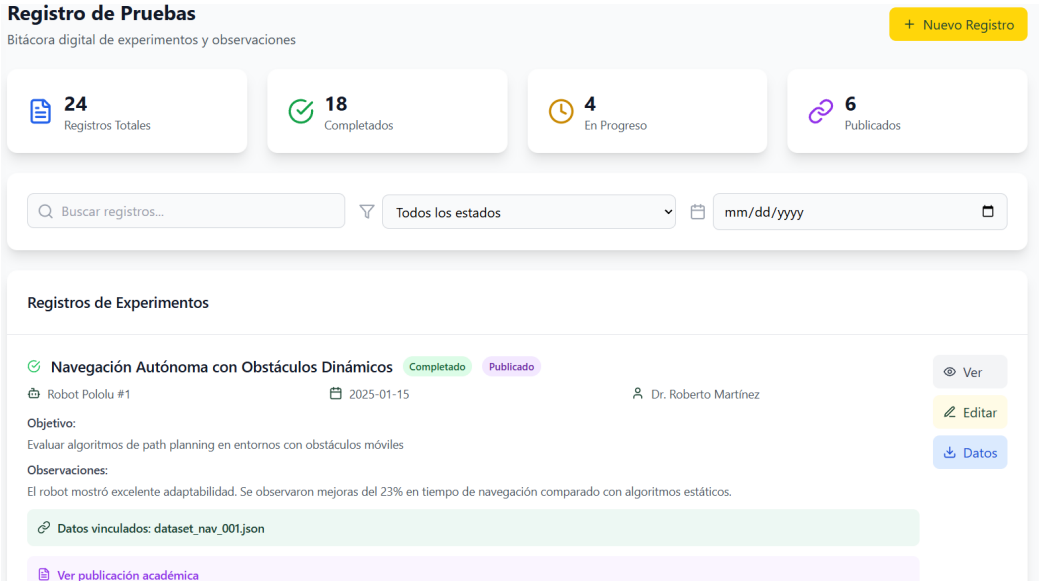
Nota. Interfaz en modo nocturno que permite al investigador monitorear experimentos, recursos y actividad del laboratorio. Elaboración propia.

Figura 25. Historial de actividad del laboratorio en modo administrador



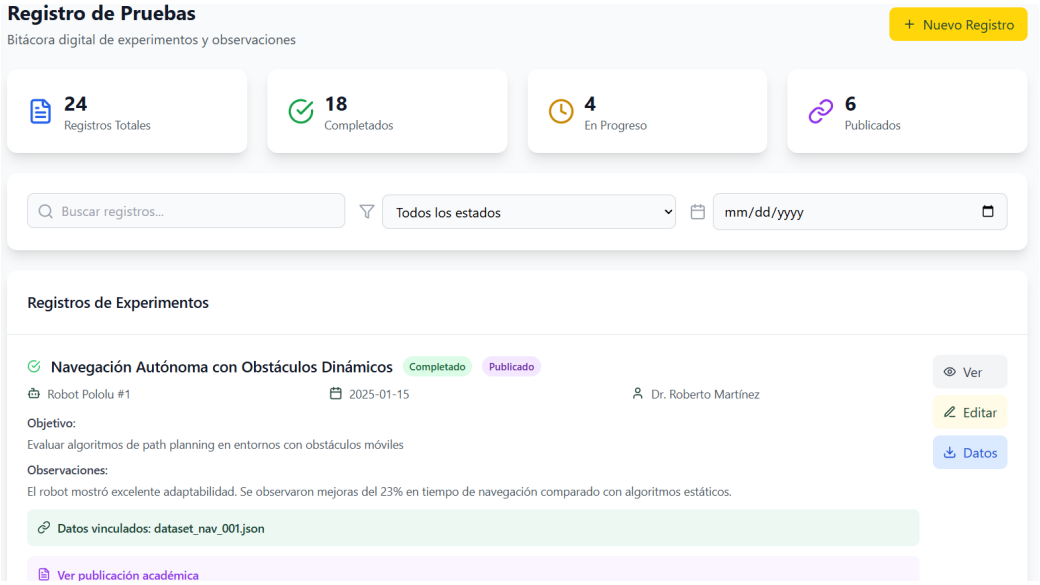
Nota. Vista del historial de actividad con métricas globales y registros individuales, disponible únicamente para administradores. Elaboración propia.

Figura 26. Vista de la bitácora digital de experimentos exclusiva de investigadores.



Nota. Ejemplo de la bitácora digital disponible solo para investigadores, donde se documentan objetivos, observaciones y resultados. Elaboración propia.

Figura 27. Panel de resultados de experimentación exclusivo del estudiantes.



Nota. Vista exclusiva para estudiantes que presenta estadísticas de experimentos realizados, incluyendo tasa de éxito, error final y tiempo de convergencia. Elaboración propia.

Repositorio con código

`https://github.com/saraxhr/Infraestructura_software_Robotat.git`

Video demostrativo control Pololu 3Pi+

`https://youtu.be/4nBqZAM2LPg`

Video demostrativo página de inicio + vista de visitante

`https://youtu.be/Ju8yoS1xv3s`

Video demostrativo vista de administrador

`https://youtu.be/3SZPo0TpwSk`

Video demostrativo vista de estudiante

`https://youtu.be/1F6UrssZe-Q`

Video demostrativo vista de investigador

`https://youtu.be/o6m4Nj3HZv8`

CAPÍTULO 16

Glosario

***backend*:** parte de un sistema de software que corre en el servidor (lado del servidor), encargada de la lógica de negocio, el acceso a datos y el procesamiento de peticiones del cliente (*frontend*). 12

***blacklist*:** también conocida como lista negra. Es un conjunto de elementos (por ejemplo, direcciones IP, usuarios o tokens) que están explícitamente prohibidos o bloqueados por no cumplir las políticas de seguridad o acceso. 30

***bundler*:** herramienta de desarrollo, especialmente en aplicaciones web, que agrupa múltiples archivos fuente (JavaScript, CSS, imágenes, etc.) y sus dependencias en uno o varios archivos optimizados (*bundles*), con el fin de reducir el número de solicitudes HTTP, mejorar la gestión de dependencias y optimizar el rendimiento de carga. 30

***frontend*:** parte de un sistema de software que se ejecuta del lado del cliente (navegador o aplicación de usuario) y que engloba la interfaz visual, controles, presentación y la interacción directa con el usuario. 32

***logs*:** registros de eventos o sucesos generados por sistemas o aplicaciones, como errores, accesos o acciones de usuarios, que permiten auditoría, diagnóstico o monitoreo. 30

API: interfaz de programación de aplicaciones. Conjunto de definiciones y protocolos que permiten que distintos componentes de software interactúen entre sí, exponiendo funciones o servicios controlados. 9