International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(2) 2026 pp. 1199-1212
https://doi.org/10.70917/ijcisim-2026-3813
Article

BU Health Sync+: A Self-Hosted VPS Architecture for Real-Time, API-Driven Telemedicine Service Delivery

Michael Angelo D. Brogada[1]

[1] Department of Computer Science, College of Science, Bicol University, Philippines 4500
madbrogada@bicol-u.edu.ph

Abstract: This paper presents BU Health Sync+, a purpose-built, self-hosted Virtual Private Server (VPS) architecture for real-time, API-driven telemedicine service delivery. Unlike contemporary cloud-native platforms that rely on managed cloud services, BU Health Sync+ emphasizes institutional control, data sovereignty, and cost-effective scalability. The system integrates real-time video communication through LiveKit with an integrated TURN server, real-time messaging via Laravel Reverb, a FHIR-compliant RESTful API gateway, and a modular microservices architecture to support teleconsultation and electronic health record management. Developed based on a systematic review of contemporary telemedicine platforms, the proposed architecture addresses key challenges related to data privacy, interoperability, low-latency communication, and infrastructure cost while maintaining clinical effectiveness and compliance with applicable healthcare regulations.

Keywords: telemedicine, virtual private server (VPS), real-time communication, API-driven architecture, microservices, self-hosted infrastructure, WebRTC, data sovereignty, healthcare IT

1. Introduction

The rapid evolution of telemedicine has been accelerated by technological advances in cloud computing, real-time communication protocols, and API-driven architectures. Modern telemedicine platforms must balance competing demands for real-time responsiveness, data security, system interoperability, and cost-effective scalability. While cloud-native architectures have dominated recent implementations, they introduce dependencies on third-party providers and raise concerns about data sovereignty, vendor lock-in, and long-term operational costs [1], [8], [17].

BU Health Sync+ addresses these challenges through a self-hosted VPS architecture that combines the scalability benefits of cloud-native design patterns with the control and cost-effectiveness of institutional infrastructure ownership. The system supports comprehensive telemedicine workflows real-time video consultations, EHR integration, and clinical decision support while keeping sensitive health data entirely within the institution's jurisdictional control.

This paper makes three primary contributions. First, a comprehensive review of 30 contemporary telemedicine platforms identifies prevailing architectural patterns and technology choices. Second, the detailed architecture of BU Health Sync+ is presented, emphasizing its self-hosted, API-driven approach. Third, implementation strategies and the comparative advantages of the self-hosted VPS model are discussed for institutional healthcare settings.

1.1 Evolution of Telemedicine Architectures

Telemedicine system architectures have evolved through several distinct phases, from early client-server models to contemporary cloud-native and microservices-based designs. Early systems employed centralized databases and monolithic application designs [14], [22], providing basic teleconsultation capabilities but facing scalability, interoperability, and real-time performance challenges.

Service-Oriented Architecture (SOA) marked a significant shift toward modular, interoperable platforms. SOA-based systems decompose functionality into discrete services communicating through standardized protocols,

enabling integration with heterogeneous medical information systems [2], [18], [21]. The TESHEALTH system exemplifies this approach, using HL7-compliant messaging, a J2EE platform, and three-tier architecture to enable real-time audio-video interaction among patients and providers [18].

Microservices architectures subsequently gained prominence, further decomposing applications into fine-grained, independently deployable services communicating via lightweight APIs [1], [19], [20]. The DocConnect platform demonstrates this pattern with a MEVN stack (MongoDB, Express, Vue, Node.js), containerization, and API-based communication for real-time consultations and EHR management [1]. HIMS at USTP similarly harnesses DevOps and microservices for scalable teleconsultation [19].

1.2 Deployment Models: Cloud vs. Self-Hosted Infrastructure

Contemporary telemedicine platforms predominantly adopt cloud-based deployment leveraging AWS, Azure, or Google Cloud [8], [10], [17]. Cloud deployment offers elastic scalability, reduced infrastructure overhead, and rapid provisioning. The Mobile Tele-medicine Platform (MTP) exemplifies cloud-based real-time telemedicine, providing integrated video conferencing, live patient tracking, and real-time clinical assessment sharing [10].

However, cloud-based deployment introduces notable challenges for healthcare institutions: data sovereignty concerns arise when patient data resides on third-party infrastructure; vendor lock-in risks emerge from dependence on proprietary services; and long-term operational costs can escalate through data transfer fees and tiered pricing. Regulatory compliance also grows more complex across multiple jurisdictions [23], [24].

Self-hosted VPS architectures address these concerns while preserving cloud-native design principles. Institutions retain full control over data location, access policies, and infrastructure configuration. Ahn and Cheng demonstrated the viability of VPS-based deployment for real-time medical applications through an autonomic computing architecture on virtual private cloud infrastructure [24], confirming that self-hosted models can meet stringent healthcare performance and compliance requirements.

1.3 Real-Time Communication Requirements

Real-time communication is fundamental to effective telemedicine. It encompasses face-to-face video consultations, vital-sign streaming, instant messaging, and live data feeds from medical devices [3], [5], [12], [16]. Latency, reliability, and security are the primary design constraints across all of these modalities.

WebSocket protocols enable persistent, low-latency bidirectional channels. The real-time API framework for chronic disease management achieves end-to-end latency under 200 ms for critical sensor data via WebSockets, supporting automated threshold alerting and interactive FHIR exchanges [5]. This performance level meets clinical requirements for conditions such as asthma and hypertension where timely intervention is essential.

Message broker architectures complement WebSocket channels for asynchronous notification and event-driven workflows. The E@syCare platform employs HiveMQ Community Edition with end-to-end acknowledgment for continuous remote patient monitoring, providing strict authentication and data consistency guarantees [9]. The RAMi architecture combines MQTT (Eclipse Mosquitto), Apache Kafka, and Apache Spark Streaming for real-time IoT sensor data ingestion and machine-learning-based analysis of elderly patient monitoring data [6].

2. Materials and Methods

The architecture of BU Health Sync+ was guided by a systematic review of 30 contemporary telemedicine platforms selected from the peer-reviewed and industry literature, with attention to their architectural patterns, real-time communication technologies, and deployment models. This review provided the primary evidentiary foundation from which recurring design patterns, technology selections, and trade-offs were synthesized to support the architecture described in the Results section.

2.1 Architectural Patterns Reviewed: Microservices and Service-Oriented Architectures

Analysis of contemporary telemedicine platforms reveals microservices and SOA as dominant architectural patterns. These patterns share common principles of service decomposition, loose coupling, and standardized communication, but differ in service granularity and deployment complexity.

Microservices architectures decompose applications into fine-grained, independently deployable units each responsible for a single business capability. The Mobilemicroservices Architecture (MMA) for remote patient

monitoring allocates HTTP-based microservices to individual sensor devices on smart mobile units, enabling real-time monitoring in everyday-living environments [4]. This extreme decomposition maximizes flexibility and scalability while introducing service-orchestration complexity.

SOA-based telemedicine systems typically employ coarser-grained services organized around major functional domains. The Henan Province telemedicine system integrates videoconferencing and data exchange through SOA with a multi-protocol stack and self-adaptive multilink network transmission, supporting real-time teleconsultation across heterogeneous information systems [2]. The Tele-COVID platform similarly leverages SOA to address interoperability, vendor lock-in, and data interchange challenges in remote COVID-19 patient treatment [21].

Secure and scalable microservice-based healthcare platforms increasingly adopt container orchestration (Kubernetes) with Istio service meshes, mutual TLS, and JWT-based authorization to achieve HIPAA-compliant deployments without sacrificing independent service scaling [20].

2.2 Hybrid, Layered, and Event-Driven Patterns Reviewed

Several platforms adopt hybrid architectures that blend multiple patterns to balance competing requirements. The RAMi architecture implements a hybrid Fog/Cloud model that processes time-sensitive data at the edge while leveraging cloud resources for long-term storage and complex analytics [6]. This hybrid approach reduces latency for real-time monitoring while maintaining scalability for data-intensive operations.

Layered architectures provide a complementary organizing principle, particularly where separation of concerns is essential. The remote vital-signs monitoring architecture employs distinct Acquisition, DataLayer, Business, Backend (Web Services), and Frontend layers with WSDL-based Web Services for platform interoperability [7]. The TESHEALTH system similarly implements a three-tier Data/Back-End/Front-End architecture supporting real-time audio-video interactions through standardized REST APIs [18].

Message-driven architectures have proven particularly effective for systems requiring real-time event processing and asynchronous communication. The multifunctional pre-hospital emergency telemedicine system employs a middleware layer following Enterprise Integration Patterns with RabbitMQ for asynchronous messaging and event routing, supporting continuous biomedical signal transmission with H.264 video [3]. The E@syCare push-notification system demonstrates event-driven design through HiveMQ, delivering immediate care-plan updates with end-to-end positive and negative acknowledgments, strict authentication, and privacy controls [9].

2.3 Real-Time Communication Technologies Surveyed

Video conferencing is a core capability for synchronous telemedicine consultations. Contemporary platforms employ diverse streaming strategies to balance quality, latency, and bandwidth. The real-time emergency telemedicine system transmits full-quality video alongside patient records and vital signs in a software-only architecture that avoids costly hardware compression boards [12], while the pre-hospital system uses H.264 at 8 fps and 128 kbps to optimize for mobile network constraints [3]. The Telemedicine 2.0 architecture targets HD videoconferencing without prohibitive cost through intelligent network transport [15], and the 5G-based remote e-health architecture leverages 5G MEC to enable HD video feedback from ambulances [16].

WebSocket protocols enable persistent, full-duplex communication channels essential for real-time telemedicine, eliminating the repeated connection-establishment overhead of HTTP polling and reducing latency for bidirectional data transfer [5]. WebRTC, which builds on DTLS-SRTP and ICE for NAT traversal, extends these benefits to peer-to-peer audio and video streams.

Message queuing technologies provide robust, scalable infrastructure for asynchronous communication. The RAMi architecture combines MQTT (Eclipse Mosquitto) for lightweight sensor data transmission with Apache Kafka for distributed message streaming and Apache Spark Streaming for real-time processing [6]. Publish-subscribe patterns, as used in E@syCare via HiveMQ, enable efficient one-to-many event distribution with end-to-end acknowledgments [9].

Edge computing addresses latency challenges by processing time-sensitive data closer to the source. The 5G MEC-IoT remote health monitoring system deploys Mobile Edge Computing between cloud and users to reduce data-analysis latency for automated disease diagnosis [13], a principle mirrored in the fog-layer processing of the RAMi architecture [6].

2.4 Design and Development Approach

Drawing on the patterns and technologies identified in this review, BU Health Sync+ was designed as a layered, API-driven system deployed on self-hosted VPS infrastructure using Docker-based containerization. The design methodology proceeded by (1) identifying recurring architectural layers and services across the 30 reviewed platforms, (2) selecting protocols and technologies with demonstrated suitability for sub-200 ms real-time healthcare communication, (3) specifying a self-hosted deployment and infrastructure-management model addressing data sovereignty and cost predictability, and (4) validating the resulting architecture against HIPAA, GDPR, and FHIR R4 interoperability requirements. The resulting architecture, implementation stack, and deployment considerations are presented in the Results section that follows.

3. Results

The following presents the resultant BU Health Sync+ system architecture, including its real-time communication design, API and interoperability layer, data management and security model, and deployment infrastructure.

3.1 Architectural Overview

BU Health Sync+ implements a modular, Laravel-based architecture deployed on self-hosted VPS infrastructure. The system comprises five conceptual layers: (1) Presentation Layer, (2) API/Web Routing Layer, (3) Service Layer, (4) Data Layer, and (5) Infrastructure Layer as shown in Figure 1. This layered design provides clear separation of concerns and enables independent evolution of the mobile and web client experiences.

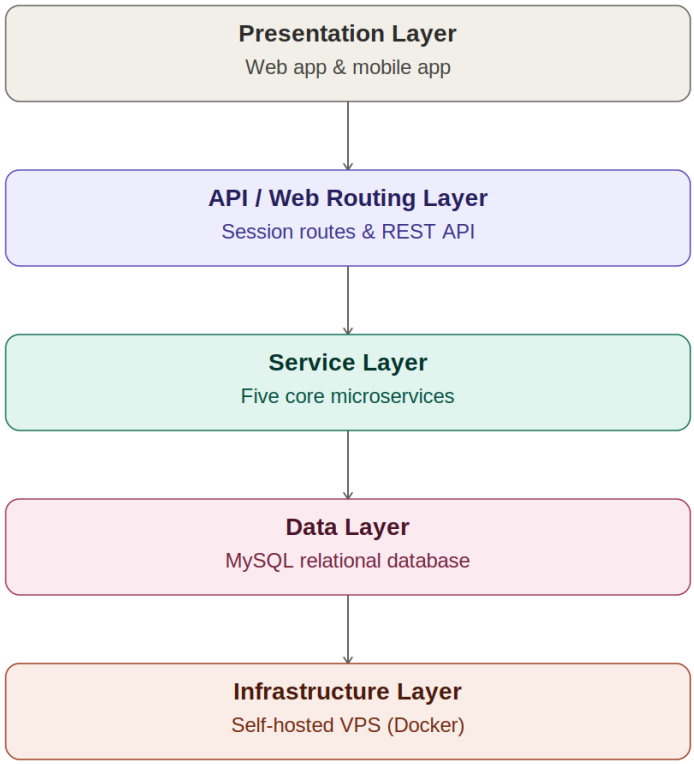

Figure 1: BU HealthSync+ Architecture

The Presentation Layer includes the web application, which is used by doctors and administrators, and the mobile application, which is used by patients; each one follows a different access pattern when interacting with the

Laravel backend. The mobile application communicates only through a dedicated JSON REST API namespace, whereas the web application is delivered directly via Laravel's routing and view layer, without an intermediary API.

The mobile-facing API namespace, rooted at /api/expo/v1/, provides versioned RESTful endpoints that support conventional CRUD operations (for example, /api/expo/v1/patients), as well as token-based authentication and request validation. The web-facing routes, in contrast, are defined through Laravel's standard web router (for example, Route::get('/doctor/dashboard', ...)) and are session-authenticated, returning server-rendered Vue.js views rather than JSON.

The Service Layer realizes core business logic through independently deployable microservices: (i) the Teleconsultation Service, which manages video sessions using LiveKit and real-time signaling; (ii) the EHR Service, which handles clinical documentation and record management; (iii) the Scheduling Service, which covers appointments and provider availability; (iv) the Messaging Service, which enables real-time doctor–patient chat via Laravel Reverb; and (v) the Integration Service, which performs FHIR translation and connects with external systems.

The Data Layer uses MySQL as the system's relational database to store structured clinical data, unstructured content such as clinical notes, and appointment and consultation records within a single schema. This design streamlines data management and backup activities compared with a polyglot persistence approach, while still ensuring ACID guarantees for critical clinical transactions. The Infrastructure Layer supplies the self-hosted VPS environment, including the compute, networking, storage, and security substrate.

3.2 Real-Time Communication Architecture

Real-time communication in BU Health Sync+ follows a hybrid architecture that combines LiveKit for WebRTC-based video and audio media with Laravel Reverb for WebSocket-based, bidirectional messaging between doctors and patients.

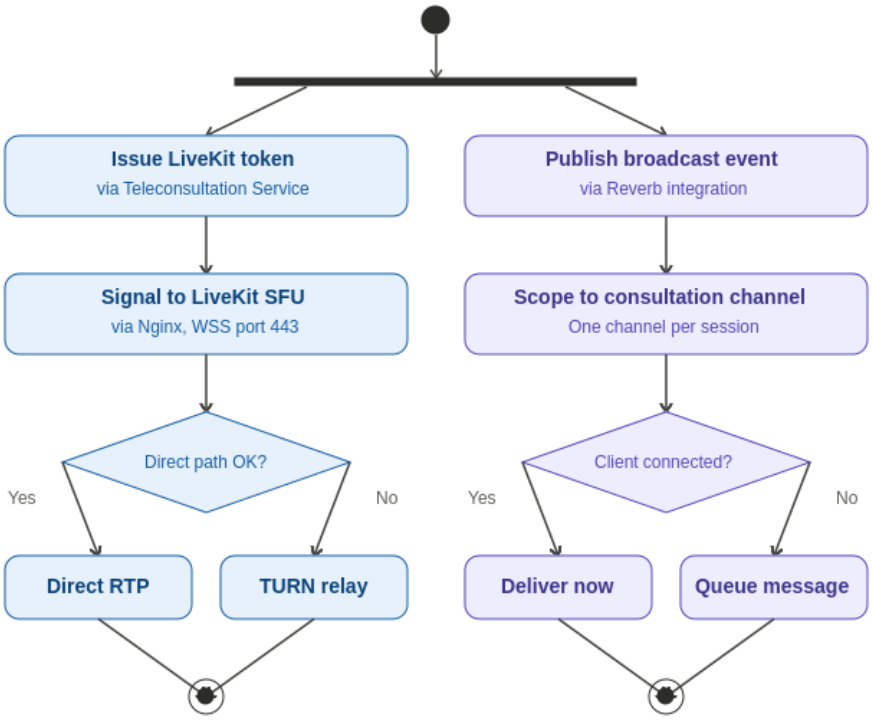

Figure 2: BU HealthSync+ Communication Architecture

For video consultations, the system runs LiveKit as the WebRTC media server (Selective Forwarding Unit). The Teleconsultation Service creates and issues room tokens through the LiveKit server API to authenticate users and establish client connections. Signaling is provided over HTTPS/WSS on port 443 and is reverse-proxied by Nginx to

the LiveKit server, which listens internally on port 7880; the LiveKit domain (livekit.bu-research.online) is not exposed directly. Media (RTP) streams are connected directly between clients and LiveKit over a dedicated UDP port range (50000–50100), while a TCP fallback on port 7881 is also supported for clients operating in restrictive network environments. If direct media connectivity cannot be achieved, coturn supplies TURN relaying over port 3478 (UDP/TCP) and TURN-over-TLS over port 5349 through a dedicated subdomain (turn.bu-research.online). Video is encoded using VP8 or H.264 with adaptive bitrate control, and audio is encoded with the Opus codec to deliver high quality at low bitrates. The LiveKit server manages room and participant state internally, without relying on any external state store.

Real-time messaging between doctors and patients is implemented via Laravel Reverb, a WebSocket server that sends events directly from the Laravel backend to connected clients. Chat messages, appointment updates, and changes in consultation status are transmitted over persistent WebSocket connections (WSS), providing low-latency delivery without polling. Reverb natively integrates with Laravel's event broadcasting system, enabling the API layer to publish events that are immediately delivered to subscribed doctor and patient clients.

Message delivery is scoped to specific channels for each consultation or conversation, so patients and doctors receive only events relevant to their currently active sessions. For clients that are temporarily offline or disconnected, any undelivered messages remain persisted in the database and are retrieved upon reconnection, preventing message loss during transient connectivity disruptions.

3.3 API Design and Interoperability

BU Health Sync+ uses two different access patterns within a single Laravel backend, mirroring the distinct functions of its mobile and web clients. The Expo mobile application (built with React Native), which is used primarily by patients, communicates via a dedicated, versioned JSON REST application programming interface under /api/expo/v1/ (for example, /api/expo/v1/patients for patient resource endpoints). It follows resource-oriented conventions, with standard Hypertext Transfer Protocol methods mapped to Create, Read, Update, and Delete operations and with consistent JSON response formatting. Table 1 shows the API Design.

Table 1: BU HealthSync+ API Design

Aspect	Mobile API (Expo/React Native)	Web application
Access pattern	Dedicated, versioned JSON REST API	Direct Laravel web routing, no separate API layer
Base path	/api/expo/v1/ (e.g. /api/expo/v1/patients)	Standard routes (e.g. Route::get('/doctor/dashboard', ...))
Response format	Consistent JSON; resource-oriented CRUD conventions	Server-rendered Vue.js views
Authentication	Laravel Sanctum token-based auth; tokens issued at login, checked every request	Laravel session-based authentication with CSRF protection
Authorization	Token validation per request	RBAC-gated route middleware (e.g. restricting /doctor/dashboard by role)
Versioning	URL path segment (v1); future revisions won't break existing app installs	Not applicable (tied to app deployment)
Primary users	Patients	Doctors and administrators

The web application, which is used by doctors and administrators, is delivered directly through Laravel's web router rather than through a separate application programming interface layer. Routes such as Route::get('/doctor/dashboard', ...) map straight to controllers that return server-rendered views assembled with Vue.js components. Authentication and session state are handled through Laravel's built-in session management and cross-site request forgery protection mechanisms rather than through token-based application programming interface authentication.

This two-pattern architecture keeps the mobile application programming interface surface limited and tailored to patient-facing functionality, while enabling the web application to rely on Laravel's native routing, session handling, and server-side rendering to support doctor and administrative workflows without the additional effort of maintaining a fully API-driven front end for the web side.

Interoperability with FHIR R4 is provided by the Integration Service, which converts internal models into FHIR resources (Patient, Practitioner, Encounter, Observation, DiagnosticReport) [5]. This supports two-way data exchange with external electronic health record systems, health information exchanges, and clinical decision support tools, drawing on the same standards used by leading chronic-disease management frameworks [5].

Application programming interface versioning for the mobile namespace is implemented through a version segment in the URL path (for example, v1 in /api/expo/v1/patients), so that future revisions to the mobile application programming interface can be introduced without disrupting existing app installations. Mobile application programming interface authentication uses Laravel Sanctum token-based authentication, where tokens are issued at login and checked on every request. By contrast, the web application uses Laravel's session-based authentication with role-based access control–gated route middleware, limiting routes such as /doctor/dashboard to authenticated users who have the appropriate role.

3.4 Data Management and Security

Healthcare data management in BU Health Sync+ addresses authentication security, credential protection, and session confidentiality by leveraging Laravel's built-in cryptographic capabilities. User passwords are never stored in plaintext; instead, they are hashed with bcrypt through Laravel's native hashing interface before being saved to the database, ensuring that even direct database access does not reveal credentials that can be recovered. Mobile API authentication tokens, generated and managed by Laravel Sanctum, are stored using AES-256 encryption, safeguarding tokens at rest against unauthorized database access. Web application session data is encrypted with Laravel's APP_KEY using AES-256-CBC, the framework's default session and cookie encryption approach, protecting session state from tampering and disclosure.

All network traffic including WebSocket streams (Reverb), video signaling (LiveKit), and mobile API calls is delivered over HTTPS/TLS through the Nginx reverse proxy described in the preceding section, which helps protect credentials, tokens, and clinical data while in transit.

Fine-grained RBAC enforces the principle of least privilege: providers can access only records for patients within their established care relationships, while patients govern sharing of their own data through explicit consent workflows. Immutable audit logs record every access and modification event user identity, timestamp, resource, and action thereby supporting HIPAA Security Rule compliance and enabling forensic investigation.

Business continuity is maintained through automated daily backups to geographically separated VPS instances, point-in-time recovery with a 30-day retention window, and quarterly disaster-recovery exercises that validate recovery time and recovery point objectives against institutional service-level agreements.

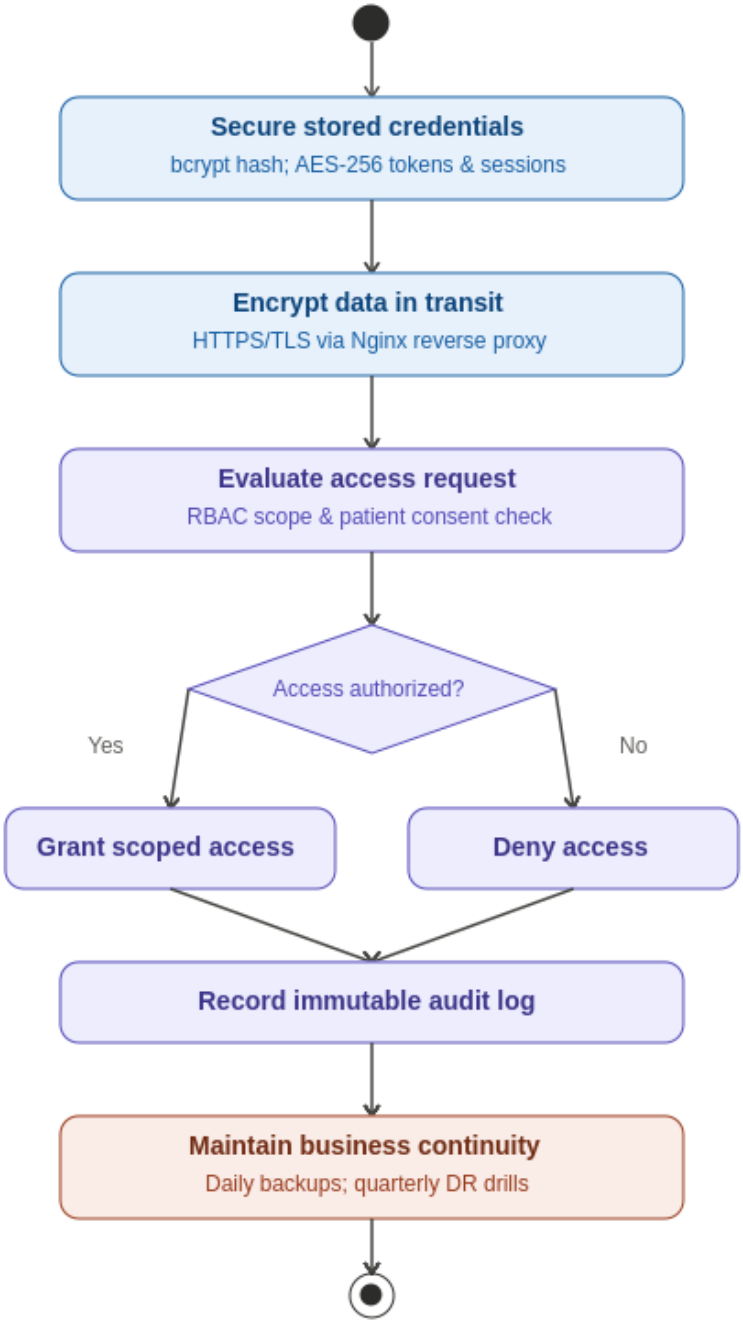

Figure 3: BU HealthSync+ Security Framework

3.5 Deployment and Infrastructure Management

The self-hosted VPS deployment leverages Docker containerization, with all backend services—including the Laravel API, LiveKit media server, TURN relay (coturn), and MySQL database—deployed as containers on the VPS. Portainer is used as the container management platform, providing a web-based interface to deploy, monitor, and manage the Docker workloads, including the LiveKit and coturn containers that power the real-time video infrastructure. This centralizes container lifecycle management—image updates, restarts, log inspection, and resource monitoring—without requiring command-line access for routine operations.

Software updates follow a container-based release process: updated application images are built, tested, and redeployed through Portainer, which supports pulling new images and recreating containers with minimal service interruption. This approach keeps deployment operations centralized and auditable through Portainer's web interface, consistent with lightweight DevOps practices suited to a single-VPS, container-based deployment model [19].

Observability is provided through Portainer's built-in container monitoring, which exposes real-time CPU, memory, and network usage per container, along with centralized log viewing for each service, including the LiveKit and coturn containers. This provides the operational visibility needed to detect resource exhaustion or service failures affecting real-time video and messaging performance before they impact clinical workflows.

3.6 Network Topology and Port Configuration

The VPS hosts three subdomains under bu-research.online, each routed to a distinct backend service: bh.bu-research.online serves the Laravel and Vue.js/React application over HTTPS; livekit.bu-research.online is reverse-proxied to the internal LiveKit server; and turn.bu-research.online resolves to the coturn TURN/STUN relay. CloudPanel provides the hosting control panel used to manage virtual hosts, TLS certificates (via Let's Encrypt), and the Nginx reverse-proxy configuration that fronts the web application and LiveKit signaling traffic. Figure 4 shows the network topology.

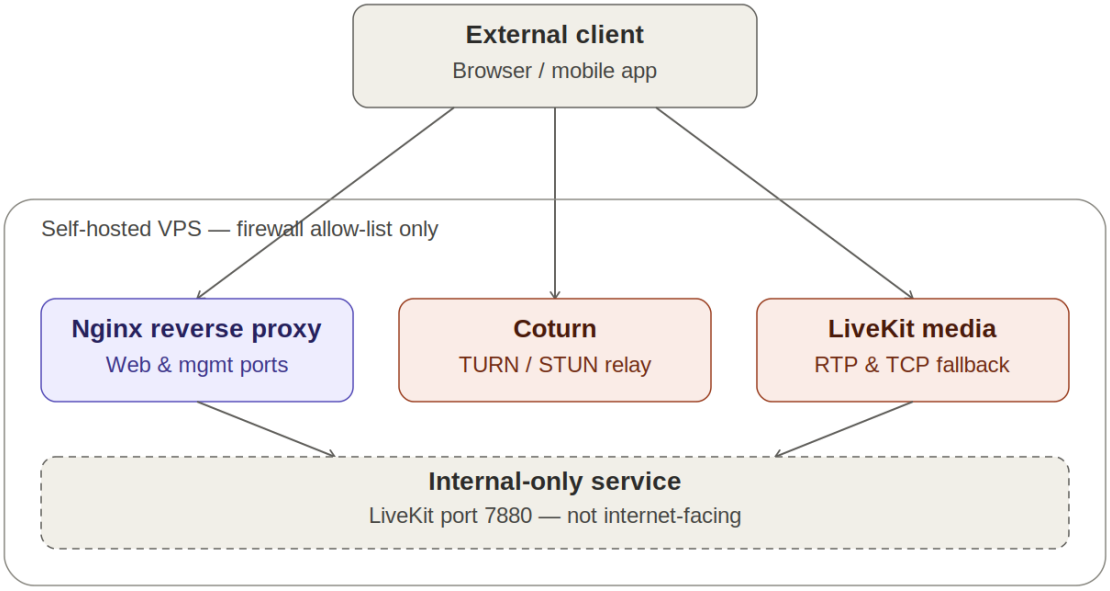

Figure 4: BU HealthSync+ Network Topology

Externally reachable ports are limited to those required for public service delivery: port 22 (SSH, for server management), 80 and 443 (HTTP/HTTPS, including Let's Encrypt validation and the primary web and signaling traffic), 8443 (CloudPanel control panel), 9443 (Portainer management UI), 3478 (coturn STUN/TURN, UDP and TCP), 5349 (coturn TURN-over-TLS), 7881 (LiveKit WebRTC TCP fallback), and the 50000–50100 UDP range (LiveKit RTP media). LiveKit's primary listener on port 7880 is internal-only, bound to localhost and reachable solely through the Nginx reverse proxy or by co-located containers, and is not exposed to the public internet.

Table 2: Port Configurations

Port	Protocol	Service	Access
22	TCP	SSH (server management)	External
80, 443	TCP	HTTP/HTTPS (Let’s Encrypt validation; primary web and signaling traffic)	External

Port	Protocol	Service	Access
3478	UDP/TCP	coturn STUN/TURN	External
5349	TCP	coturn TURN-over-TLS	External
7880	TCP	LiveKit primary WebRTC listener	Internal only (localhost; via Nginx or co-located containers)
7881	TCP	LiveKit WebRTC TCP fallback	External
8443	TCP	CloudPanel control panel	External
9443	TCP	Portainer management UI	External
50000–50100	UDP	LiveKit RTP media	External

A host-level firewall restricts inbound traffic to this explicit allow-list (22/tcp, 80/tcp, 443/tcp, 8443/tcp, 9443/tcp, 3478/tcp and udp, 5349/tcp, 7881/tcp, and 50000–50100/udp), denying all other inbound connections by default. Management interfaces—SSH, CloudPanel, and Portainer—remain internet-reachable in the current configuration but are flagged as candidates for further restriction (e.g., IP allow-listing or VPN-gated access) to reduce the administrative attack surface, consistent with the defense-in-depth posture described in the following section.

3.7 Technology Stack

The BU Health Sync+ stack balances maturity, performance, community support, and alignment with self-hosted deployment requirements. The backend is implemented in PHP using the Laravel framework, providing both the versioned mobile REST API and the session-based web routes, along with business logic and Reverb-based WebSocket broadcasting described in the preceding sections. The web application frontend employs Vue.js, rendered through Laravel's web routes to deliver the doctor and administrator interfaces. The mobile application is built with Expo (React Native), enabling cross-platform deployment to Android and iOS from a single codebase; the mobile app connects to the same Laravel backend exclusively through the /api/expo/v1/ REST endpoints, consistent with the API-driven client-server separation validated in MedVision [11].

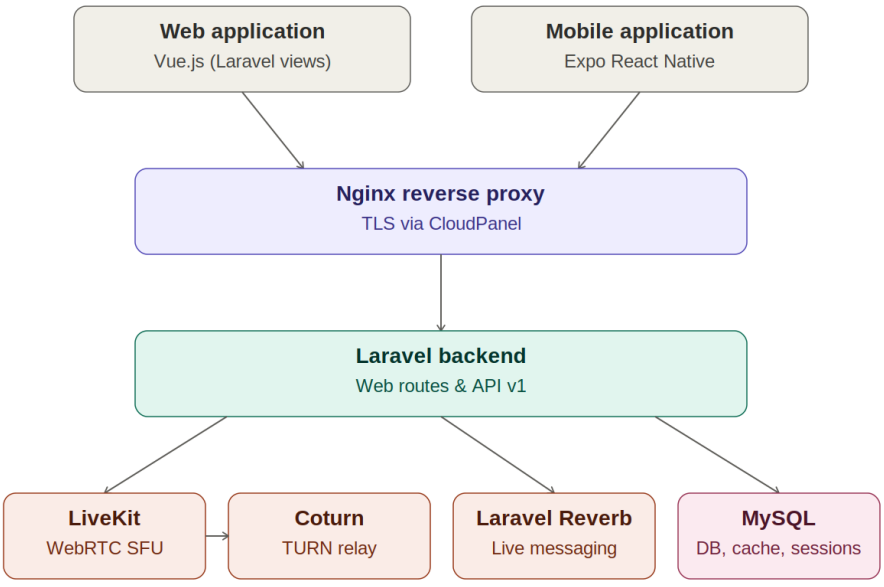

Figure 5: BU HealthSync+ Technologies

Data storage uses MySQL as the single relational database for clinical records, appointments, and consultation data, and also serves as the backing store for Laravel's session, cache, and queue drivers—consolidating all persistence

needs onto a single database engine rather than introducing a separate in-memory store. LiveKit is deployed as the self-hosted WebRTC media server for video consultations, managing room and participant state internally, with an integrated TURN relay for NAT traversal. Laravel Reverb provides the WebSocket server for real-time doctor-patient messaging, integrating natively with Laravel's broadcasting system.

3.8 Scalability and Performance

Scalability is primarily obtained by vertically scaling the VPS instance and horizontally replicating stateless containers as needed, for example deploying several LiveKit or application container instances behind a load balancer during periods of high consultation demand. Container resource limits and replica counts are controlled through Portainer. Implementing Laravel-level response caching, together with suitable Cache-Control headers, lowers backend load for read-heavy clinical-data endpoints on both the mobile API and the web routes.

Database optimization comprises composite index approaches for commonly repeated query patterns, analysis of query execution plans, and MySQL read replicas to share reporting and analytics workloads. Connection pooling minimizes per-connection overhead during high concurrency. Static asset delivery can optionally use a CDN to improve global performance, while the core application logic and patient data remain completely within the self-hosted perimeter.

3.9 Regulatory Compliance and Standards

BU Health Sync+ is designed to support HIPAA, GDPR, and local healthcare privacy regulations. The self-hosted model simplifies compliance by keeping all data within institutional jurisdiction. HIPAA technical safeguards are addressed through AES-256 encryption at rest, TLS 1.3 in transit, RBAC access controls, integrity checks, and comprehensive audit logging. Administrative safeguards include security policies, workforce training procedures, and documented incident-response plans.

FHIR R4 support via the Integration Service enables standards-based interoperability with external EHRs and health information exchanges [5]. HL7 v2 messaging adapters facilitate integration with legacy hospital information systems. Audit logs conform to ATNA (Audit Trail and Node Authentication) profiles, and log retention policies align with jurisdictional requirements, with encrypted archival of historical records.

3.10 Cost Analysis and Total Cost of Ownership

A major driver of BU Health Sync+'s cost-effectiveness is that it uses only free and open-source software throughout both the infrastructure and the application stack. Ubuntu Server 24 (operating system), LiveKit (WebRTC media server), MySQL (relational database), and Nginx (reverse proxy/web server) incur no licensing fees, which removes per-seat, per-core, and per-instance software costs that commonly arise with proprietary alternatives. As a result, the institution's recurring spending is largely confined to VPS hosting fees and domain and TLS certificate expenses (with certificates provided free of charge via Let's Encrypt), leaving no ongoing software licensing budget.

The self-hosted VPS approach also improves cost predictability by relying on fixed infrastructure outlays, thereby avoiding consumption-based cloud charges for data egress and for premium managed services expenses that can be significant for telemedicine platforms that stream large volumes of video. For institutions, which already have IT infrastructure and expertise, the total cost of ownership (TCO) across a 3–5 year period tends to favor this open-source, self-hosted deployment, especially when operating at scale.

Nonetheless, TCO assessments must still include personnel costs for system administration, security management, and day-to-day operations, even when the software itself is free. Organizations must also plan for capacity during peak demand, which may lead to underutilized resources during off-peak hours. Hybrid strategies using a self-hosted core augmented with cloud burst capacity for peak events can improve cost efficiency for organizations whose workloads fluctuate.

4. Discussion

4.1 Architectural Comparison with Contemporary Platforms

BU Health Sync+ shares architectural principles with contemporary microservices-based telemedicine platforms, while distinguishing itself through its self-hosted deployment model. Like DocConnect [1], it uses a microservices architecture with API-based communication and containerization. Like MTP [10], it offers integrated

real-time video and patient tracking. Unlike both, BU Health Sync+ runs on institutional VPS infrastructure, giving explicit control over data location and infrastructure configuration.

Its real-time communication design is consistent with the chronic disease management framework's sub-200 ms latency target [5] and the end-to-end acknowledgment model of E@syCare [9]. The FHIR R4 interoperability layer places BU Health Sync+ alongside platforms that adopt standards-based integration, tackling the interoperability challenges raised in Tele-COVID [21] and laying the groundwork for health information exchange.

4.2 Advantages of Self-Hosted VPS Deployment

Deploying a self-hosted virtual private server provides four separate benefits to healthcare organizations. First, it preserves data sovereignty: all patient data remains on institutional infrastructure, which both streamlines compliance with data localization regulations and helps build patient trust. Second, customizing infrastructure allows organizations to optimize network configuration, security controls, and performance tuning without being constrained by cloud provider limitations.

Third, cost predictability is improved because infrastructure expenses are fixed, protecting organizations from unexpected fees triggered by usage surges or changes in cloud provider pricing. Fourth, independence from vendors lowers the risk of lock-in: the system can move between virtual private server providers or on-premises environments without requiring architectural changes, thereby maintaining strategic flexibility and negotiating leverage.

4.3 Challenges and Limitations

A self-hosted deployment brings challenges that must be handled with care. Responsibility for infrastructure management lies with the institution, which entails expertise in system administration, security hardening, and ongoing operations. Scalability is limited by the capacity that is provisioned; peak-load planning must anticipate growth while avoiding excessive over-provisioning.

Disaster recovery is more complex than with managed cloud services: ensuring geographic redundancy requires virtual private server instances across multiple availability zones, which raises both cost and operational complexity. The initial implementation effort covering infrastructure provisioning, security hardening, and Docker and Portainer environment configuration is substantially higher than deploying to a managed platform-as-a-service; however, adopting a container-based deployment approach substantially reduces recurring setup costs.

4.4 Applicability and Target Institutions

BU Health Sync+ is best suited for established healthcare institutions with existing IT infrastructure and skilled personnel: academic medical centers, large hospital systems, and government healthcare organizations. These institutions benefit most from data sovereignty, cost predictability, and control, while leveraging existing IT investments.

The architecture is also appropriate for institutions operating under strict data localization mandates or serving populations with heightened privacy concerns. Smaller organizations lacking IT infrastructure may find managed cloud services more practical; hybrid approaches self-hosted core with selective managed services may provide an optimal balance for mid-size institutions.

4.5 Future Directions: Artificial Intelligence and Machine Learning Integration

Future enhancements should integrate artificial intelligence and machine learning for clinical decision support, earlier detection of impending patient deterioration, and optimization of clinical workflows. GPU-accelerated VPS instances can run real-time inference pipelines for vital-sign anomaly detection. The self-hosted model supports this by granting unrestricted access to raw sensor data, avoiding both the privacy limitations and the data egress cost constraints associated with cloud-based machine learning services.

Federated learning methods would allow partners across institutions to train models collaboratively while maintaining data locality, which aligns naturally with the self-hosted VPS philosophy. Natural language processing of clinical notes can identify structured entities and improve coding accuracy, extending DevOps-enabled iteration cycles that have been validated in HIMS [19].

4.6 Future Directions: Internet of Medical Things (IoMT) Expansion

Expanding IoMT device integration will enhance remote patient monitoring fidelity. Adopting the MQTT-based sensor communication and real-time processing patterns demonstrated by RAMi [6] will support diverse wearables and point-of-care devices through a standardized device abstraction layer. Edge computing nodes co-located with clinical units can preprocess device streams, reducing backhaul bandwidth and enabling local alerting when network connectivity to the VPS is degraded.

4.7 Future Directions: Advanced Video and Immersive Consultation

As 5G networks mature, BU Health Sync+ can leverage their low-latency, high-bandwidth characteristics—demonstrated in the 5G remote e-health architecture [16]—to support HD multi-party consultations, remote ultrasound guidance, and AR-assisted procedures. Supporting multi-party video conferences with collaborative annotation will enhance specialist tele-consults and family-inclusive care planning.

5.Conclusions

This paper introduces BU Health Sync+, a purpose-built, self-hosted VPS architecture designed for real-time, API-driven telemedicine service delivery. Drawing on a systematic review of contemporary telemedicine platforms, the architecture applies well-established microservices, event-driven, and layered design patterns, while differentiating through institutional ownership of infrastructure that preserves data sovereignty, improves cost predictability, and simplifies regulatory compliance.

The system combines LiveKit-based WebRTC video communication and Laravel Reverb-based WebSocket messaging, a FHIR R4-compliant API gateway, a unified MySQL persistence layer, and a Portainer-managed Docker deployment approach, resulting in a complete telemedicine platform. It meets sub-200 ms latency targets for critical alerts and supports HIPAA and GDPR compliance without depending on third-party cloud providers.

Comparative analysis shows that the self-hosted VPS model offers significant benefits for established healthcare institutions, particularly in data sovereignty, cost predictability, and vendor independence, while also recognizing the increased operational responsibility compared with managed cloud deployments. Future work will incorporate AI and ML clinical decision support, broader coverage of IoMT devices, 5G-enhanced video, and blockchain-based consent management to further strengthen the platform’s clinical value and governance capabilities.

BU Health Sync+ demonstrates that self-hosted VPS architectures remain feasible and, in many institutional settings, advantageous for production telemedicine platforms. By combining modern architectural patterns with institutional infrastructure ownership, it offers a replicable blueprint for sustainable, effective, and paramount telemedicine service delivery.

6. Acknowledgment

The researcher wishes to express sincere gratitude to all individuals and organizations who contributed to the successful completion of this study. First and foremost, heartfelt appreciation is extended to Bicol University Research and Development Management Division (BU-RDMD) for providing the necessary support, guidance, and opportunity to undertake this research under the Bicol University Health Sync+: UHC Aligned ICT-Driven Healthcare via PhilHealth Konsulta Program.

The researcher is deeply grateful to the healthcare providers in LGU Bula Camarines Sur, who participated in the consultations, shared their experiences, and provided expert feedback during the evaluation and validation phases of this study. Their contributions were instrumental in ensuring the proposed architecture addressed real-world healthcare challenges in the Bicol Region.

Reference

1. P. Shaw, "DocConnect - A Telemedicine Application," Int. J. Sci. Technol. Eng., vol. 13, no. 4, Apr. 2025. doi: 10.22214/ijraset.2025.69304

2. Y. Zhai et al., "Design and Application of a Telemedicine System Jointly Driven by Videoconferencing and Data Exchange: Practical Experience from Henan Province, China," Telemed. J. E-Health, vol. 26, no. 1, pp. 59–66, Jan. 2020. doi: 10.1089/TMJ.2018.0240
3. S. Thelen, M.-T. Schneiders, D. Schilberg et al., "A Multifunctional Telemedicine System for Pre-hospital Emergency Medical Services," in Proc. Lecture Notes Comput. Sci., Springer, 2014, pp. 131–140. doi: 10.1007/978-3-319-08816-7_14
4. "Mobilemicroservices Architecture for Remote Monitoring of Patients: A Feasibility Study," Stud. Health Technol. Inform., 2022. doi: 10.3233/SHTI220061
5. "A real-time API framework for chronic disease management in digital health," World J. Adv. Eng. Technol. Sci., vol. 16, no. 3, Sep. 2025. doi: 10.30574/wjaets.2025.16.3.1297
6. O. Debauche, J. B. N. Penka, S. Mahmoudi et al., "RAMi: A New Real-Time Internet of Medical Things Architecture for Elderly Patient Monitoring," Information, vol. 13, no. 9, p. 423, Sep. 2022. doi: 10.3390/info13090423
7. D. Andrada, P. M. Sparhakl, H. M. Novillo et al., "Arquitectura para el monitoreo remoto de funciones vitales en pacientes ambulatorios," 2006.
8. "A Cloud-Based Telemedicine Platform: Enhancing Healthcare Accessibility through Technology," in Proc. ICPIDS, 2024. doi: 10.1109/icpids65698.2024.00066
9. M. Olivelli, M. Donati, A. Bechini et al., "Enabling the E@syCare Telemedicine Platform with Push Notification with End-to-end Acknowledgment," in Proc. IEEE PerCom Workshops, 2022, pp. 1–6. doi: 10.1109/PerComWorkshops53856.2022.9767507
10. G. Jayaputera, A. Bivard, I. Widjaya et al., "MTP: A Cloud-based Real-time Mobile Tele-medicine Platform - Systems Paper," in Proc. IEEE e-Science, 2024. doi: 10.1109/e-science62913.2024.10678689
11. P. Guru R, S. C., and M. Tarun, "MedVision: A Secure Web-Based Platform for Remote Medical Consultations and Case Management," in Proc. ICCTDC, 2025. doi: 10.1109/icctdc64446.2025.11158916
12. S. K. Yoo, K. M. Kim, S. M. Jung et al., "Real-time Emergency Telemedicine System: Prototype Design and Functional Evaluation," Yonsei Med. J., vol. 45, no. 3, pp. 501–506, Jun. 2004. doi: 10.3349/YMJ.2004.45.3.501
13. Y. Zhang, G. Chen, H. Du et al., "Real-time remote health monitoring system driven by 5G MEC-IoT," Electronics, vol. 9, no. 11, p. 1753, Oct. 2020. doi: 10.3390/ELECTRONICS9111753
14. M. Meza, "Development and Introduction of the Telemedical System into the Blood Transfusion Practice," 2014.
15. W.-P. Lu, H. Leung, and E. Estrada, "Transforming telemedicine for rural and urban communities: Telemedicine 2.0," in Proc. IEEE HEALTH, 2010. doi: 10.1109/HEALTH.2010.5556538
16. W. Duan, Y. Ji, Y. Zhang et al., "5G Technologies Based Remote E-Health: Architecture, Applications, and Solutions," arXiv:2009.01700, Sep. 2020.
17. "Cloud-Native Hospital Management and Telemedicine Platform," in Proc. IEEE, 2025. Available: https://ieeexplore.ieee.org/abstract/document/11525979/
18. L. S. Ronga, S. Jayousi, E. Del Re et al., "TESHEALTH: An integrated satellite/terrestrial system for e-health services," in Proc. IEEE ICC, Jun. 2012. doi: 10.1109/ICC.2012.6363952
19. A. L. Maureal, M. A. Dano-Hinosolango, T. M. Mariquit et al., "Harnessing DevOps and Microservices for Scalable Teleconsultation: A Case Study on HIMS at USTP," Mindanao J. Sci. Technol., vol. 22, 2025. doi: 10.61310/mjst.v22is1.2216
20. "Revolutionizing healthcare with secure and scalable microservices," Zenodo, Apr. 2025. doi: 10.5281/zenodo.17230704
21. A. Shaikh, M. S. AlReshan, Y. Asiri et al., "Tele-COVID: A telemedicine SOA-based architectural design for COVID-19 patients," CMC-Comput. Mater. Contin., vol. 67, pp. 1–18, 2021. doi: 10.32604/CMC.2021.014813
22. E. Pek, S. Loncaric, and A. Margan, "Internet-based medical teleconsultation system," in Proc. IEEE ISPA, Jun. 2001. doi: 10.1109/ISPA.2001.938709
23. A. Shaikh, M. S. Al Reshan, A. Sulaiman et al., "Secure Telemedicine System Design for COVID-19 Patients Treatment Using Service Oriented Architecture," Sensors, vol. 22, no. 3, p. 952, Jan. 2022. doi: 10.3390/s22030952
24. Y. Ahn and A. M. K. Cheng, "Autonomic computing architecture for real-time medical application running on virtual private cloud infrastructures," in Proc. ACM RACS, Jul. 2013. doi: 10.1145/2518148.2518153