UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

Tema:

"SISTEMA DE TELEMEDICINA CON MONITOREO DE SIGNOS VITALES BASADO EN IOT EN UN AMBIENTE SMART TV"

Trabajo de Titulación Modalidad: Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Electrónica y Comunicaciones

ÁREA: Física y Electrónica

LÍNEA DE INVESTIGACIÓN: Tecnología de la información y Sistemas de Control.

AUTOR: Roberto Andrés Garcés Salazar

TUTOR: Ing. Santiago Manzano, Mg.

Ambato - Ecuador

Marzo – 2021

APROBACIÓN DEL TUTOR

En calidad de tutor del Trabajo de Titulación con el tema: "SISTEMA DE TELEMEDICINA CON MONITOREO DE SIGNOS VITALES BASADO EN IOT EN UN AMBIENTE SMART TV", desarrollado bajo la modalidad Proyecto de Titulación por el señor Roberto Andrés Garcés Salazar, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 15 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y el numeral 7.4 del respectivo instructivo.

Ambato, marzo 2021.

Firmado electrónicamente por:
VICTOR SANTIAGO MANZANO VILLAFUERTE

Ing. Santiago Manzano, Mg.
TUTOR

AUTORÍA

El presente Proyecto de Investigación titulado: "SISTEMA DE TELEMEDICINA CON MONITOREO DE SIGNOS VITALES BASADO EN IOT EN UN AMBIENTE SMART TV" es absolutamente original, auténtico y personal. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, marzo 2021.

Roberto Andrés Garcés Salazar

C.C. 1804626065

AUTOR

APROBACIÓN TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Titulación presentado por el señor Roberto Andrés Garcés Salazar, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Titulación, titulado "SISTEMA DE TELEMEDICINA CON MONITOREO DE SIGNOS VITALES BASADO EN IOT EN UN AMBIENTE SMART TV" nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 17 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora presidenta del Tribunal.

Ambato, marzo 2021.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

Ing. Pilar Urrutia, Mg.
PRESIDENTA DEL TRIBUNAL

Firmado electrónicamente por:
FREDDY GEOVANNY BENALCAZAR PALACIOS

Dr. Freddy Benalcázar, Mg.
PROFESOR CALIFICADOR

Firmado electrónicamente por:
MARIO GEOVANNY GARCIA CARRILLO

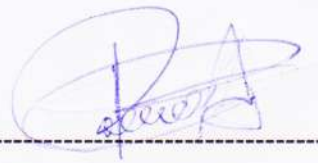
Ing. Mario García, Mg.
PROFESOR CALIFICADOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación en favor de la Universidad Técnica de Ambato, con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la institución.

Ambato, marzo 2021.



Roberto Andrés Garcés Salazar

C.C. 1804626065

AUTOR

AGRADECIMIENTO

A mi hermano Jonathan por su total apoyo en la realización de mi proyecto de titulación, a mis padres por su aliento económico y emocional, a mis compañeros de estudios, que me ayudaron a superar cada etapa, y a "la femme de ma vie", por su apoyo emocional e incondicional.

INDICE GENERAL

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN EJECUTIVO

La actual pandemia ha dificultado la atención médica y el acceso a los distintos servicios de salud convencionales, que en ocasiones pueden ser de difícil acceso para personas que habitan lejos del servicio de salud más cercano, o no pueden movilizarse debido al padecimiento de distintas afecciones médicas.

En este proyecto se implementa un sistema de telemedicina con una arquitectura enfocada en el hogar utilizando IoT basada en cuatro elementos: Interfaz de usuario, administración, conectividad y sensores médicos. En la capa de interfaz de usuario se desarrolló una aplicación con el framework React-Native para el sistema Android TV, la cual recibirá los datos sensados de la capa de sensores médicos del sensor de BPMs y SpO2 MAX30100 a través de bluetooth, embebido dentro de un circuito electrónico Wearable. Estos datos serán enviados a través del protocolo TCP/IP y administrados mediante Odoo, alojado en una instancia EC2 de Amazon. La información almacenada puede ser desplegada a través de tablas y gráficas, alertar al usuario al recibir datos anormales, y enviar prescripciones a la aplicación de Android TV oportunamente.

Se realizó una encuesta a cuarenta profesionales de la salud con el fin de conocer el beneficio que aporta el sistema propuesto. De esta muestra, 88,5 por ciento estaría dispuesto a implementar el sistema en su lugar de trabajo. Además, debo mencionar que este trabajo forma parte del proyecto de investigación “Sistema de Telemedicina para la monitorización de señales vitales en un ambiente de Smart TV”, aprobado mediante la Resolución Nro. UTA- CONIN-2020-0297-R.

Palabras clave: Telemedicina, IoT, Weareable, nube.

ABSTRACT

The current pandemic has hampered medical care and access to the various conventional health services, which can sometimes be difficult for people living far from the nearest health service, or they cannot be mobilized due to various medical conditions.

This project implements a telemedicine system with a home-focused architecture using IoT based on four elements: User interface, administration, connectivity, and medical sensors. In the user interface layer, an application was developed with the React-Native framework for the Android TV system, which will receive the sensing data from the medical sensor layer of the BPMs and SpO2 MAX30100 sensor via Bluetooth, embedded within a wearable electronic circuit. This data will be sent via the TCP/IP protocol and managed via Odoo, hosted in an Amazon EC2 instance. The stored information can be displayed through tables and graphs, alert the user when receiving abnormal data, and send prescriptions to the Android TV app in a timely manner.

A survey of forty health professionals was carried out to ascertain the benefit of the proposed system. Of this sample, 88.5 percent would be willing to implement the system at their workplace. In addition, I should mention that this work is part of the research project "Telemedicine System for the Monitoring of Vital Signals in a Smart TV Environment", approved by Resolution No. UTA- CONIN-2020-0297-R.

Keywords: Telemedicine, IoT, Weareable, cloud.

CAPÍTULO I

MARCO TEORICO

1.1 Antecedentes investigativos

El avance e implementación de dispositivos IoT (Internet of things) han permitido convertir nuestros objetos cotidianos en objetos inteligentes. Además de la automatización de dispositivos y objetos que se pueden controlar a distancia, cuando es aplicado a la salud permite que muchas personas, sin importar las largas distancia que deben recorrer para tener acceso a los sistemas de salud convencionales, puedan aprovechar los servicios de salud sin la necesidad de moverse de su hogar. Gracias a la IoT y las tecnologías que permiten el despliegue de la telemedicina, los sistemas convencionales de salud que actualmente se conocen como "hospital-céntrico" se transformaran en sistemas de salud centrados en el hogar, el cual favorece a la carga laboral de los trabajadores de salud, reducción de costos operacionales, así como la reducción de contagios en casos de riesgos biológicos (como el actual covid-19) [1].

La aplicación de las telecomunicaciones ha ayudado a desplegar atención médica a personas en lugares de difícil acceso, o que se han establecido en lugares remotos. Aunque se ha desarrollado en su mayor parte en países de Europa y Norte América, en América Latina su impacto sería mayor, debido a las características geográficas y socioeconómicas [2].

En Ecuador, la empresa Saludsa ha desplegado un sistema de telemedicina disponible exclusivamente para sus clientes, el cual explica que el 70% de las consultas médicas son tratables a distancia. El sistema puede emitir diagnósticos, ordenes de exámenes y prescripciones médicas a través de correo electrónico una vez finalizada la consulta con un médico [3]. Paralelamente la Universidad de San Francisco de Quito implementó un proyecto de telemedicina para apoyar al Ministerio de Salud desde el año 2015, y al sistema de salud del Seguro Social desde el año 2019. El sistema se basa en dar segundas opiniones en casos médicos atendidos por médicos de Galápagos y de la Amazonia. En este caso, la telemedicina no se realiza en tiempo real, ya que

los médicos de la red de la Universidad colocan sus colaboraciones en una plataforma informática en un periodo de 24 horas [4].

En el sector médico varios sensores electrónicos se utilizan para recolectar signos vitales necesarios para monitorear y recolectar información útil para atender afecciones y realizar diagnósticos a tiempo. Los sensores del Internet de las cosas (IoT) se pueden automatizar para recolectar información y enviarla mediante una conexión Wireless a un profesional médico para su evaluación. Esto ayuda a un diagnóstico oportuno y efectivo. Al mismo tiempo, el despliegue exitoso del IoT para el cuidado de la salud puede verse afectado por el uso efectivo de diferentes tipos de sensores que monitorean al paciente, la eficiencia de los dispositivos IoT en consumo energético y costos, la integración de varios tipos de datos, conexión con la nube, algoritmos eficientes para el análisis de datos e interfaces amigables para los usuarios [5].

La realización de este proyecto interesa tanto al sector médico como al sector de las telecomunicaciones y de la electrónica, cuyo objetivo es la telemedicina aplicando la tecnología IoT. Además del sector informático interesado en el diseño de "Front-End" y "Back-End" de gestión y almacenamiento de datos alojados en la nube. Este trabajo da un aporte teórico y práctico al proyecto de investigación titulado "**Sistema de Telemedicina para la monitorización de señales vitales en un ambiente de Smart TV**", aprobado mediante la Resolución Nro. UTA- CONIN-2020-0297-R.

Ya que el proyecto se basa en una tecnología estudiada a nivel mundial y que se ha desarrollado extensamente, se han podido inquirir las siguientes investigaciones acerca del tema propuesto y que han aportado a su desarrollo.

En el año 2016, Carlos Rivas Costa y su equipo implementa y prueba una solución para proporcionar servicios sociales y de salud para ancianos en el hogar basados en tecnologías de Smart TV. Las aplicaciones van desde interacción con redes sociales hasta monitoreo de signos vitales; desde juegos interactivos de televisión hasta aplicaciones convencionales de atención en línea, como recordatorios de medicamentos o telemedicina [6]. Este proyecto aporta un concepto claro de un ambiente Smart TV desarrollado mediante software libre, una estructura para un sistema de monitorización de signos vitales almacenadas en bases de datos y que se pueden consultar mediante servicios conocidos como "*Backend*".

En el año 2018, se implementa una solución de telesalud basada en IoT diseñada específicamente para abordar las necesidades de las personas mayores. El usuario final interactúa con un televisor para registrar parámetros biométricos y recibir advertencias y recomendaciones relacionadas con grabaciones de sensores ambientales y de salud. VITABOX, el componente central del sistema se instaló en un entorno de laboratorio, simulando un escenario real y probado por varias personas [7]. Este proyecto aporta una estructura para el registro y monitorización de parámetros médicos a través de un sistema embebido de TV y que son sensados utilizando dispositivos Weareables que se comunican por medio de Bluetooth. Además, se detalla un servicio de API REST para la consulta de historiales de sus datos médicos.

En el año 2018, Najan M. Abdelsamee y Abeer Algarni desarrollan una aplicación móvil para el intercambio de datos entre enfermos cardíacos y sus consultores para ayudar en el diagnóstico y monitoreo rápidos. La aplicación aporta con el diseño de una aplicación basada en la nube y consta de dos partes que incluyen al cliente y servidor. El módulo del servidor se ejecuta en el lado del servidor que reside en una nube de Microsoft Windows Azure. Este módulo se ejecuta todo el tiempo y está a la espera de recibir un mensaje del cliente o un médico general [8]. Este proyecto aporta un sistema de almacenamiento de datos médicos para el envío de prescripciones una vez que el profesional ha evaluado los datos, donde está siempre a la espera de recibir alguna prescripción.

En el año 2018**,** B. Himanshu y sus colegas desarrollan un sistema que incluye la recolección y evaluación continua de múltiples signos vitales, atención médica a largo plazo y una conexión celular a un centro médico en caso de emergencia, transfiriendo todos los datos sin procesar por Internet. Se desarrolló e implementó con éxito un prototipo de dicho sistema, que ofrece un alto nivel de atención médica con una importante reducción de costos. Mediante el uso de este sistema, los médicos pueden comunicarse fácilmente con otros galenos y forenses con experiencia [9]. Este proyecto aporta en la programación de un microprocesador para capturar los datos de los sensores y procesarlos, para luego poder ser enviados por una conexión TCP/IP donde son almacenados en bases de datos para que después puedan ser analizados.

En el año 2019, L. Jian y sus colegas presentan un sistema en el cual se monitorea la frecuencia cardíaca, la adquisición de señales mioeléctricas y un sistema de

monitorización del flujo sanguíneo, en personas mayores solitarias. Las señales fisiológicas se toman del lado del paciente y se presentan en una tableta portátil o teléfono móvil. En esta investigación se utiliza tecnología RFID y sensores para identificar y monitorear automáticamente la actividad humana. El sistema mide la frecuencia del pulso a partir de la periodicidad de la señal, en función de los cambios en el flujo sanguíneo en tiempos determinados. Además, el sistema informa a los familiares del paciente a través de la nube en situaciones de riesgo [10]. Esta investigación aporta en la programación de un Arduino para adquirir y procesar los datos obtenidos por los sensores médicos colocados en el cuerpo del paciente, para luego ser almacenados en bases de datos que son consultadas por los profesionales de la salud.

1.1.1. Fundamentación teórica.

1.1.1.1. Ambientes Smart TV.

Introducción

Con la llegada del internet y las conexiones de alta velocidad como la fibra óptica, se han desarrollado nuevas alternativas para consumir contenido multimedia. La televisión cambio de su habitual modo de visualizar canales con contenido específico y programado de baja definición, a contenido bajo demanda de alta definición y hasta 4K, donde el usuario puede acceder al contenido sin necesidad de ajustarse a un horario definido por las televisoras, dando lugar al termino Smart TV o televisión inteligente [11, p. 6].

Debido al crecimiento de la Smart TV y de las múltiples posibilidades de desarrollo de aplicaciones para el entretenimiento y multimedia, esta se ha vuelto una plataforma vital para el hogar. En esencia un ambiente Smart TV permite la interacción con una señal de TV, acceso a internet, transmisión en vivo, publicidad e incluso aplicaciones para el diario vivir. A diferencia de la tecnología IPTV (Televisión por Protocolo de Internet), la tecnología Smart TV permite el acceso a múltiples funciones y aplicaciones por medio de un sistema operativo y plataformas de software, permitiendo no solo transmitir audio y video sino también múltiples aplicaciones útiles [12].

Definición.

Un ambiente SMRT TV se define generalmente como un medio que proporciona transmisión de TV, Internet, aplicaciones y servicios inteligentes a través del montaje de una CPU y plataforma operativa en el decodificador o pantalla [12], permitiendo a los espectadores acceder convenientemente a diferentes contenidos multimedia y servicios interactivos en una sola plataforma. Un ambiente Smart TV también ofrece servicios interactivos basados en Internet, incluyendo "Media-on-demand", redes sociales y juegos en línea [13].

Software

Los softwares más comunes están basados en Linux, y en algunos casos, basados en el paradigma "web-como-plataforma" como HTML5, CSS o JavaScript. Al momento de esta investigación se han encontrar los siguientes tipos de software utilizados en Smart TV [14, p. 8].

- *Android TV:* Aunque Android TV puede parecer nuevo en el mercado, tiene una larga historia. Se anunció por primera vez como Google TV en Google I / O 2010. Los primeros dispositivos disponibles fueron de Logitech y Sony. Dos años más tarde, en Google I / O 2014, se anunció un nuevo dispositivo basado en Android: Android TV. La programación de aplicaciones para Android TV sigue la misma pauta que la de un smartphone, facilitando su desarrollo [15, pp. 173-174].
- *WebOS:* es un sistema operativo basado en Linux desarrollado por Palm. El desarrolla para WebOS está basado en HTML5 (Lenguaje de Marcas de Hipertexto versión 5), JavaScript y CSS (Hojas de Estilo en Cascada), permitiendo el acceso de aplicaciones web a las funcionalidades nativas del sistema en donde está alojado. Las aplicaciones en WebOS están compuestas de código escrito en HTML5, en conjunto con métodos en JavaScript [14, p. 9].
- *Apple TV:* es un software para televisores inteligentes creado por Apple Inc. En este caso, la distribución de las aplicaciones se basa en la adquisición por medio del Apple Store [14, p. 10]. Las herramientas de desarrollo son similares a las del sistema iOS. Se pueden utilizar plantillas de TVMLKit (Lenguaje de Marcas para TV) escritas en XML (Lenguaje de Marcas Extensivo) y

JavaScript, que a su vez permiten utilizar API (Interfaz de Programación de Aplicaciones) de JavaScript para crear aplicaciones [16].

- *Tizen OS:* es un sistema operativo flexible construido desde cero para satisfacer con las necesidades de los desarrolladores de dispositivos móviles, constructores de dispositivos, operadores móviles, desarrolladores de aplicaciones y proveedores de software independiente (ISV). Tizen además ofrece el desarrollo de aplicaciones nativas con soporte HTML5 [17].
- *FireOS:* es un sistema operativo que se ejecuta en Fire Tv y tabletas de Amazon. Fire OS es una derivación de Android, por lo que una aplicación que se ejecuta en Android es probable que se ejecute también en dispositivos Fire de Amazon. Por ejemplo, Fire OS está basado en Android 9.0 [18].
- *Firefox OS:* es un sistema operativo de código abierto basado en Linux y en la tecnología Mozilla Gecko. Fue diseñado inicialmente para dispositivos móviles, pero algunos constructores de Smart TV como Panasonic han aplicado este sistema operativo como software en sus televisores. Las aplicaciones se desarrollan en HTML5, CSS y JavaScript como si fuera una página web [14, pp. 9-10].

De todos estos sistemas operativos, Android TV presenta mayor sencillez al momento de desarrollar aplicaciones [14, p. 12], además que se puede encontrar mayor soporte de desarrollo en las páginas oficinales de Android. Asimismo, según un estudio publicado en el año 2020 en la página web “Statista”, Android TV abarca aproximadamente el 40% del mercado a nivel mundial, liderando como el sistema operativo más utilizado en el año 2018 [19].

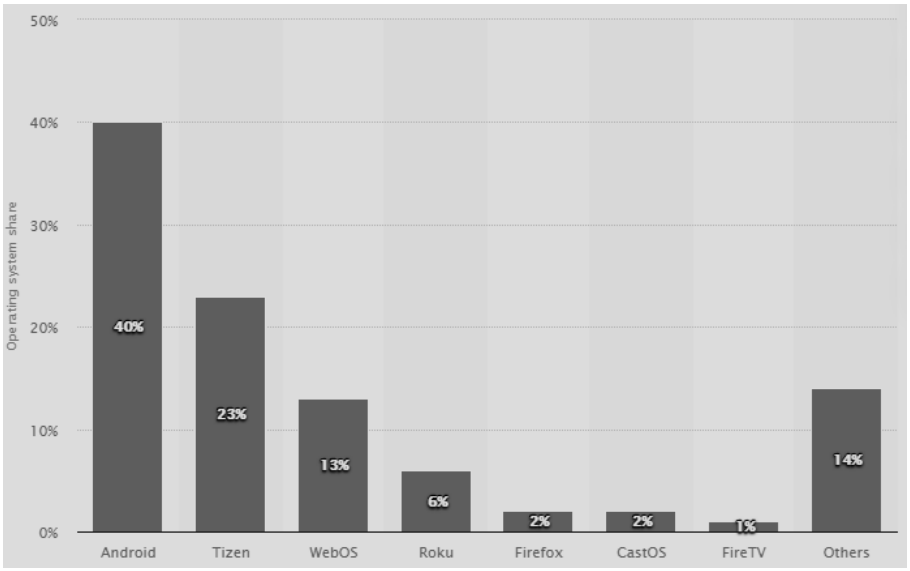


Fig. 1. Estudio de los sistemas operativos más utilizados en el 2018 a nivel mundial [19].

Android TV

Android es un sistema operativo de código abierto basado en un kernel de Linux. Como se muestra en la Fig. 2, Android está compuesto de cuatro capas: el kernel de Linux, librerías, framework de aplicaciones y aplicaciones [20, p. 5].

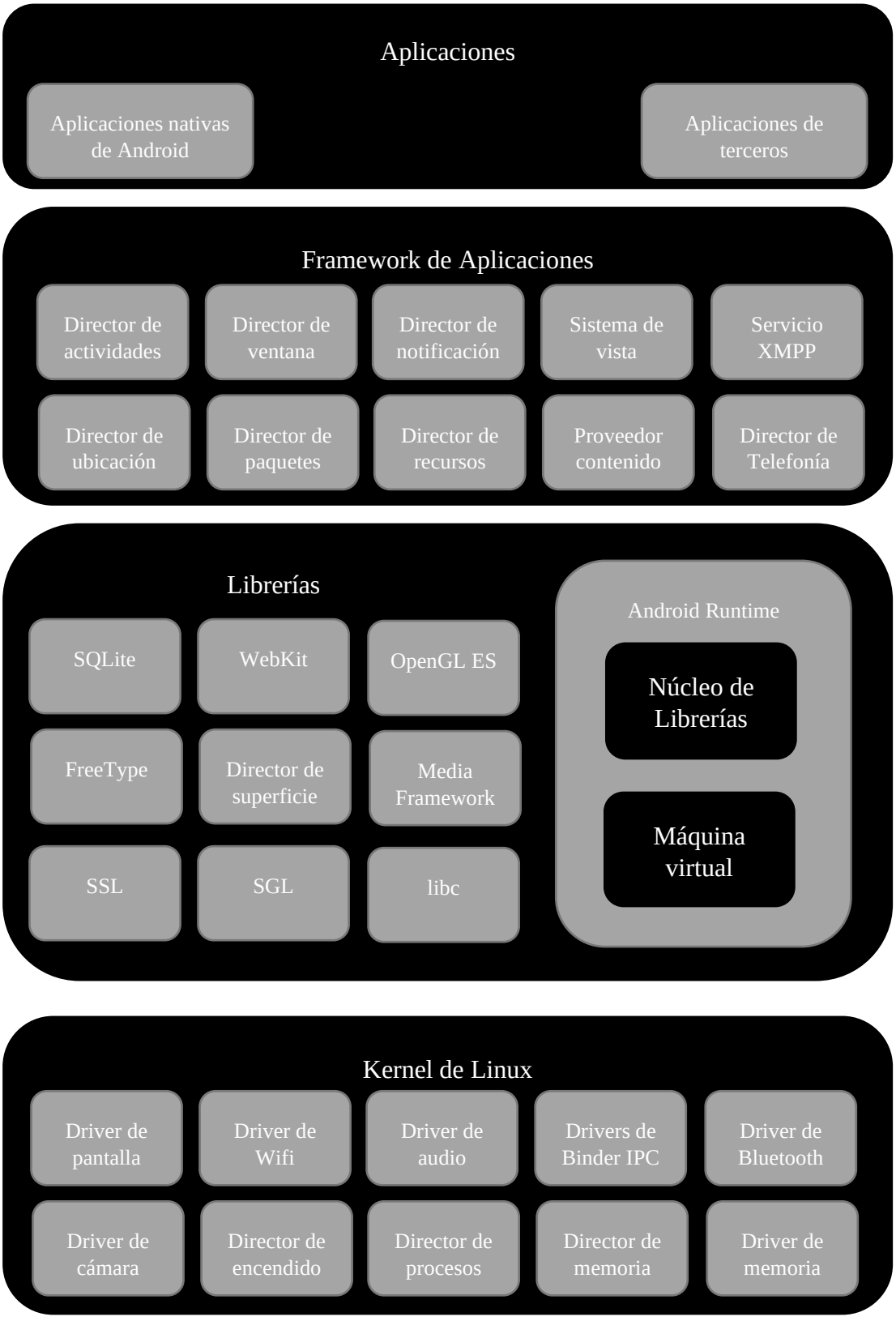


Fig. 2. Arquitectura del sistema operativo Android [11, p. 16].

- *Kernel de Linux:* El kernel de Linux es una base de roca sólida para la ejecución del código computacional. Este provee varias funcionalidades como el manejo de la memoria, manejo de los periféricos (como la cámara, los botones de bloqueo y volumen, etc.). El software de Android interactúa con el hardware por medio de esta capa, además de contener los controladores de los periféricos del dispositivo [21, p. 279].
- *Capa de librerías:* En esta capa superior al kernel de Linux se encuentran las librerías nativas de Android, la cual permite que el dispositivo pueda manipular diferentes tipos de funciones [21, p. 279], como las que permiten el desarrollo de interfaces para el usuario o el acceso a bases de datos (SQLite) [11, p. 16].
- *Android run time:* Android run time permite ejecutar las aplicaciones, a la vez que transforma el código de bytes, compilándolo en instrucciones nativas que se ejecutan en un entorno de ejecución [20, p. 7]. De esta forma, cada aplicación de Android se ejecuta dentro de su propia instancia de entorno de ejecución ART (Android run time) [11, p. 16].
- *Framework de Aplicaciones:* Se le considera un conjunto de servicios que conforman un entorno donde se gestión y ejecutan las aplicaciones de Android. Entre sus funciones se encuentran las notificaciones, la localización, intercambio de contenido entre aplicaciones, y el control del ciclo de vida de alguna aplicación [11, pp. 16-17]. Además, este proporciona acceso a servicios de programación de un nivel superior, programadas mediante clases Java, como son las APIs [20, p. 7].
- *Aplicaciones:* La capa de aplicaciones es la capa superior de la arquitectura y es donde las aplicaciones se alojan. Estas pueden ser preinstaladas de fábrica, permitiendo a los usuarios navegar por internet o leer correo electrónico [20, p. 7]. Así mismo, estas aplicaciones pueden ser instaladas por el usuario, de fuentes de terceros, permitiendo añadir múltiples funcionalidades de software en el sistema operativo [11, p. 17].

Hardware

Las Smart TV son computadores embebidos muy parecidos a los teléfonos inteligentes, en los que se pueden integrar múltiples periféricos de entrada de un típico

computador como teclado, ratón, sistemas de sonido de alta definición, cámaras de video, etcétera [22, p. 4].

Paralelamente a los Smart TV existen los conocidos TV-box, que son sistemas embebidos con los mismos periféricos que un Smart TV. Estos permiten convertir a un televisor ordinario, con una entrada de audio y video, en un televisor inteligente. La mayor parte de estos contienen el sistema operativo Android TV, pero actualmente la empresa Apple Inc. ha desarrollado su propia versión con Apple TV [14, pp. 11-12].

A continuación, se presenta una relación entre los fabricantes de Smart TV más conocidos y el sistema operativo con el que trabaja.

TABLA I. Relación entre fabricante y el sistema operativo utilizado [14, p. 11].

Fabricante	Sistema Operativo
LG	WebOS
Panasonic	Firefox OS
Samsung	Tizen
SONY	Android TV

Según el periódico "EL COMERCIO" los televisores con mayor popularidad en el Ecuador son las marcas LG y SONY [23]. Para este proyecto es necesaria una conexión bluetooth nativa, y para ello se han investigado diferentes marcas y fabricantes disponibles en el mercado ecuatoriano, que se muestran a continuación.

TABLA II. Comparación de los distintos tv-box disponibles en el mercado ecuatoriano.

Elaborado por el autor.

Marca	Sistema operativo	Wifi	Bluetooth nativo	Memoria RAM	Memoria Interna	Procesador	Precio
SpeedBOX fusion	Android TV SpeedBoxUI 7.1.2	IEEE 802.11 b/g/n 2.4GHz	Si	2 GB DDR3	16 GB	Amlogic Quad-Core 2.0GHz 64 bit	149.00
Xiaomi Mi Box	Android TV 9.0	IEEE 802.11 b/g/n 2.4GHz	No	2 GB DDR3	8 GB	Amlogic S905X-H Quad-core Cortex-A53 2.0GHz 32 bit	84.99
Mecool KM1	Android TV 9.0	IEEE 802.11 b/g/n/ac 5-2.4GHz	No	2 GB DDR3	16 GB	Amlogic S905X3 Quad-core Cortex-A53 2.0GHz 32 bit	79.99
Tanix	Android TV 10	IEEE 802.11 b/g/n/ac 5-2.4GHz	No	2 GB DDR3	16 GB	Allwinner H6, Quad-Core	64.95
X96 Max	Android TV 9.0	IEEE 802.11 b/g/n/ac 5-2.4GHz	No	4 GB DDR3	64 GB	S905X3 Quad Core 1.9 GHz 32 bit	64.99

1.1.1.2. Entornos de desarrollo

En algunos casos la creación de aplicaciones para sistemas operativos de Smart TV, siguen las mismas pautas que para la creación de aplicaciones para teléfonos inteligentes, con ligeros cambios pensados para ejecutarse en una pantalla de televisor, como es el caso de Android TV y Apple TV [14, p. 14]. La programación de aplicaciones se puede llevar a cabo de tres formas diferentes:

- *Aplicaciones nativas:* están son desarrolladas para ser alojadas directamente dentro del sistema operativo del cliente. Estas solo se pueden encontrar directamente en las tiendas de aplicaciones dedicadas como lo son Google Play en el caso de Android TV, y Apple Store en el caso de Apple TV. Los lenguajes de programación y las herramientas son específicas para cada sistema, por ejemplo, para desarrollar una aplicación Android TV, el desarrollador puede utilizar el entorno React-Native, que se basa en el SDK de Android de java. Así como para Apple TV, por medio de la herramienta XCode [24]. Aunque React-Native permite crear aplicativos iOS desde una plataforma basada en el mismo sistema. Las aplicaciones nativas están más centradas para la creación de contenido, gracias al acceso al hardware del dispositivo en el cual se aloja, permitiendo acceder a todas sus funcionalidades [25].
- *Aplicaciones multiplataforma:* el concepto de multiplataforma nace de permitir a los desarrolladores crear aplicaciones para varias plataformas en un solo paso, evitando la reincidencia en el desarrollo y aumentado su producción. Muchas de estas se desarrollan gracias a las tecnologías Web como HTML, CSS y JavaScript [26], siendo lenguajes intermediarios que no se alojan directamente en el dispositivo. Estos deben ser utilizados en dispositivos donde existan bajos recursos en rendimiento [14, p. 15].
- *Aplicaciones web hibridas:* están son aplicaciones multiplataforma que se interpretan en la interfaz del dispositivo utilizando un navegador web embebido. Estas utilizan las mismas tecnologías que el anterior, que son HTML, CSS y JavaScript. Es una buena opción cuando se necesita que la aplicación corra en varios sistemas operativos y dispositivos. Al igual que el anterior, este no se aloja en el dispositivo, provocando tiempos de respuesta reducidos, cuando los niveles de rendimiento son bajos [14, p. 15].

React

React es un framework de JavaScript, creado por los desarrolladores de Facebook en el año 2013 para resolver problemas desarrollando complejas interfaces de usuarios cuando los datasets (conjunto de datos) cambian todo el tiempo [27, p. 1]. React brinda el lado "vista" del paradigma de desarrollo "modelo-vista-controlador" o MVC, representando la V del MVC. Este puede trabajar tanto del lado del cliente como del lado del servidor, dando como resultado una comunicación interoperable entre los dos [20, p. 8]. React funciona gracias a dos principios:

- *Virtual DOM:* quizás este sea el principio más importante en el funcionamiento de React. Cuando una página web se carga en un explorador, un DOM (Modelo de Objetos del Documento) se crea, el cual contiene la página web. Un DOM es una representación jerárquica de una página web, mostrando su estado actual por medio de código escrito en HTML. Cada vez que el usuario interactúa con la página web, como dirigirse a una ventana o recibir información de la nube, el contenido del DOM es reestructurado por medio de JavaScript [20, p. 8].
- *Componentes y JSX:* JSX es la capa que transforma el código XML para escribir componentes React, en un código que React utiliza para renderizar código en JavaScript. Esto no es necesario para escribir código en React, pero es altamente recomendable ya que hace que la ejecución del código sea más suave [27, p. 17]. Por otro lado, los componentes permiten crear la UI (Interfaz de Usuario) de las aplicaciones. Estos son parecidos a las funciones, que contienen parámetros y propiedades. Estos pueden ser creados gracias a código JavaScript o JSX. Además, en React se puede crear tags HTML como cadenas de texto o componentes de React [20, p. 9].

React Native

Facebook lanzo este framework de programación en el año 2015. React Native combina las mejores partes de la programación en React y las librerías de mejor clase de JavaScript para realizar interfaces de usuarios [28]. La principal ventaja de React Native es el desarrollo multiplataforma, permitiendo escribir código de funcionalidades similares a través de las plataformas, ya que las interfaces graficas

pueden variar según las funcionalidades del sistema en el que se está ejecutando [20, p. 10]. A continuación, se presenta un cuadro comparativo entre el principal desarrollador para Android "Android Studio" y React Native.

TABLA III. React Native vs Android Studio.

Elaborado por el autor.

Característica	React Native	Android Studio
Desarrollo	Permite crear varias aplicaciones para diferentes plataformas a la vez, ahorrando tiempo en el desarrollo.	Desarrollo de una aplicación exclusiva de Android.
Lenguaje	JavaScript.	Java, Kotlin.
Soporte	Soporte a varias plataformas al mismo tiempo.	Soporte solo para Android.

Los componentes de React renderiza el código nativo existente interactuando con las APIs nativas que otras aplicaciones utilizan, a través del paradigma de Interfaz de Usuario declarativo de React Native y JavaScript [28]. En la Fig. 3 se representa la diferencia entre la renderización en React y React Native.

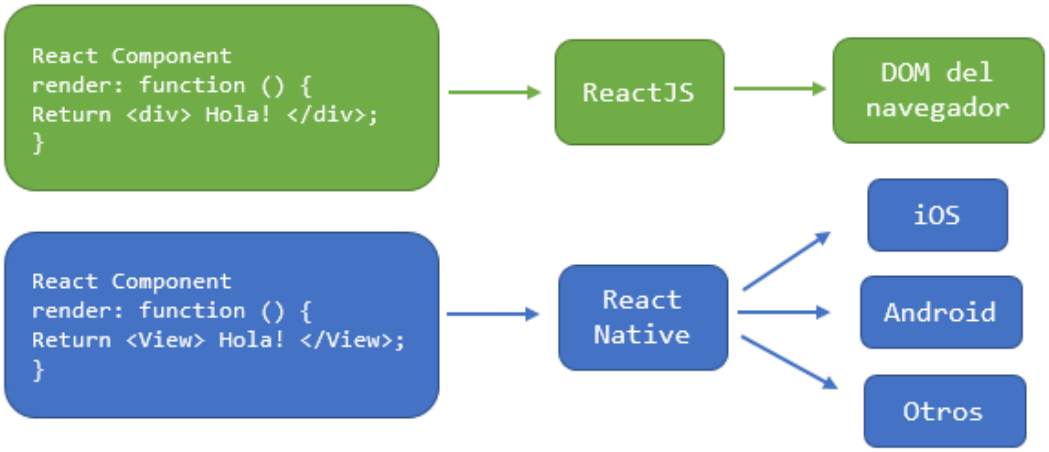


Fig. 3. Renderización en React y React Native [20, p. 10].

El soporte para dispositivos de televisión inteligentes fue implementado por React Native para que las aplicaciones ya existentes puedan renderizarse con pocos o ningún cambio en el código de JavaScript. Uno de los más importantes se realiza en el archivo "AndroidManifest.xml". Este archivo contiene información previa a la ejecución de alguna aplicación, como son los servicios de broadcast, el paquete de java donde se aísla la aplicación, los permisos de acceso a las APIs nativas y el nivel de API para la ejecución de la aplicación [11, p. 19]. Específicamente en React Native se debe agregar o modificar las líneas de código que se muestran en la Fig. 4.

```
<!-- Add custom banner image to display as Android TV launcher icon -->
<application
 ...
 android:banner="@drawable/tv_banner"
 >
   ...
   <intent-filter>
     ...
     <!-- Needed to properly create a launch intent when running on Android TV -->
     <category android:name="android.intent.category.LEANBACK_LAUNCHER"/>
   </intent-filter>
   ...
 </application>
```

Fig. 4. Modificaciones al archivo "AndroidManifest.xml" para la ejecución de aplicaciones React Native en dispositivos Android TV [28].

1.1.1.3. IoT (Internet de las cosas)

Introducción

El internet de las cosas (IoT-Internet of things) nos ayuda a recolectar información del medio mediante dispositivos con sensores embebidos que son capaces de conectarse al internet, facilitando el almacenamiento en la nube para su posterior análisis y toma de decisiones [29, p. 42]. En todo caso, el fin del IoT es el de conectar diferentes cosas para conseguir un objetivo, y tales pueden ser los Smart TV. Estos dispositivos, como muchos otros sistemas de computación embebida, poseen ya sea un puerto de conexión Ethernet o una tarjeta de red Wifi, permitiéndoles interactuar con múltiples servicios que se encuentran en internet, así como las APIs, bases de datos, servicios de streaming, etc. Características que permiten a dichos dispositivos el ser adaptado en un proyecto IoT [30, p. 163].

Dispositivos y arquitectura

IoT se trata de monitorización remota, tecnología, y también en donde se aplican las mismas. IoT también puede centrase en las tecnologías que tienen un papel esencial en la innovación y en el procesamiento avanzado y complejo en entornos confinados y cerrados, como son la automatización industrial (Industria 4.0) [31, pp. 14-15]. En la Fig. 5 se puede observar una arquitectura global del IoT, tanto sus aplicaciones más comunes y los sitios en donde se pueden desplegar sistemas de este tipo.

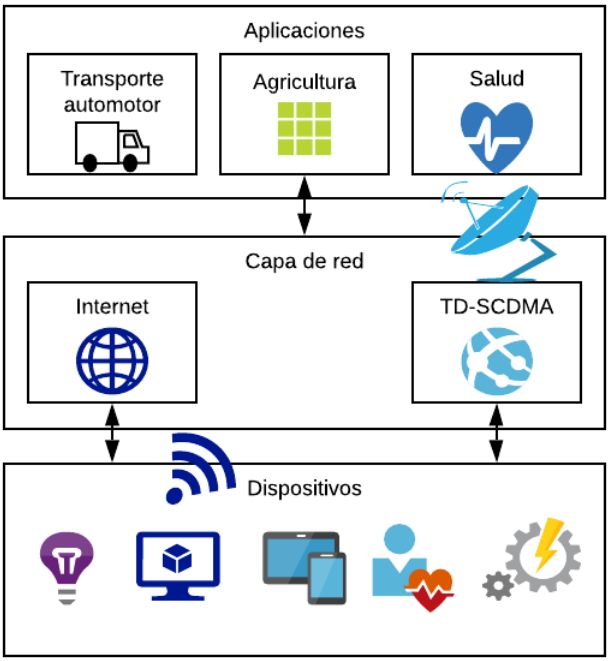


Fig. 5. Arquitectura del Internet de las cosas.
Desarrollado por el investigador.

- *Aplicaciones:* las aplicaciones están conformadas por las distintas áreas en donde el Internet de las Cosas puede ser implementado, rigiendo un procesamiento especifico de los datos de acuerdo con el tipo de aplicación. Las aplicaciones pueden tener varios objetivos como la seguridad, la convivencia, reducción de costos, la optimización de procesos, o en algunos casos, ayudar en el diario vivir en el hogar o exteriores [31, p. 16].
- *Capa de red:* en esta sección se encuentra todo aquel dispositivo que permite la comunicación entre los dispositivos de captura de datos y las aplicaciones, estos pueden ser puertas de enlaces que manejan los diferentes estándares de

comunicación, como son el IEEE 802.11g o el IEEE 802.16 para los dispositivos, y TCP/IP o HTTP para las aplicaciones. Estos estándares pueden variar de acuerdo con la aplicación [31, p. 87].

- *Dispositivos:* estos son los sensores o actuadores que interactúan con el usuario, brindando información del mundo real hacia los dispositivos de la capa de red, que después los pasaran a la aplicación para su procesamiento o análisis. Estos se pueden comunicar con los dispositivos de enlace mediante comunicación LAN o WLAN, con la única condición de manejar el mismo tipo de estándar de comunicación [31, p. 84].

Aplicaciones del IoT

Existe un sin número de aplicaciones en donde el IoT puede ser desplegado, siendo cada uno de ellos tan importante como el anterior. En la tabla 5 se pueden listar varios ejemplos de aplicaciones de acuerdo con tendencias e intereses de distintos sectores. Como se puede observar, existen aplicativos desde gadgets para el hogar, hasta la reforestación de plantas y árboles [31, p. 16].

TABLA IV. Aplicaciones del IoT y sus objetivos [31, p. 16].

Aplicación	Objetivo
Artículos electrónicos	• Gadgets para el hogar • Robótica • Weareables • Sensado participativo • Web social de las cosas
Transporte automotor	• Vehículos autónomos • Transporte multimodal
Banca Comercial	• Micro pagos • Logística minorista • Información de vida útil de productos. • Asistencia de compras
Medioambiental	• Contaminación del aire, agua, suelos • Clima • Contaminación por ruido
Infraestructuras	• Edificios y hogares autónomos • Carreteras, ferrocarriles
Servicios básicos	• Redes energéticas inteligentes

	• Manejo del agua • Manejo de gas, aceite y energías renovables • Manejo de desperdicios • Calefacción
Salud y bienestar	• Monitoreo remoto • Vida asistida • Cambio de comportamiento • Cumplimiento de tratamientos • Deportes y fitness
Ciudades inteligentes	• Entornos integrados • Optimización de operaciones • Convivencia • Socioeconomía • Sustentabilidad • Vida inclusiva
Procesos industriales	• Robótica • Fabricación • Manejo de recursos naturales • Operaciones remotas • Automatización • Manejo de maquinaria pesada
Agricultura	• Reforestación • Cultivos y agricultura • Agricultura urbanística • Control de ganado y pesca

Estándares

El internet de las cosas está presente en muchas de las áreas y campos de las tecnologías, para facilitar el manejo y dar soporte mediante el uso de varios protocolos y estándares de comunicación. El Consorcio World Wide Web (W3C), Grupo de trabajo de ingeniería de Internet (IETF) y el Instituto de ingenieros eléctricos y electrónicos (IEEE) han propuesto la utilización de varios estándares de comunicación en el Internet de las Cosas, permitiendo su utilización en varios campos y aplicaciones, además del despliegue físico en edificios, hogares, plantaciones, cría de animales, etcétera [32, p. 484]. La tabla 6 muestra un resumen de las funciones, elementos y el nivel del internet de las cosas, especificando los estándares y protocolos que se usan en cada uno de los niveles.

TABLA V. Estándares según el nivel, función y elementos del IoT [32, p. 485].

Nivel	Función	Elementos
Sensado	Medir y sensar datos físicos.	Sensores inteligentes, sensores embebidos, dispositivos RFID, actuadores.
Comunicación	Brinda los protocolos para los enlaces y puertas de acceso a nivel infraestructural y aplicativo.	**Nivel infraestructural:** LTE-A, Z-wave, IEEE 802.15.4, EPC-global, RPL, 6LowPAN, IPV4/IPV6.
		Nivel Aplicativo: MQTT, CoAP, XMPP, DDS, AMQP, HTTP-REST.
Computacional	Provee de procesamiento y almacenamiento en el nivel de software y hardware.	**Hardware:** Dispositivos programables como microcontroladores, Microprocesadores, dispositivos inteligentes, Arduino, Raspberry-Pi, etc.
		Software: TinyOS, RaspbianOS, RiotOS, Android, etc.
Análisis	Computación en la nube y normalización para aplicaciones específicas.	Agregación de la información, servicios de investigación colaborativa, servicios omnipresentes.

Tecnologías Inalámbricas

Así como el incremento del número de dispositivos inteligentes, las aplicaciones del IoT siguen el mismo curso. Muchas soluciones IoT necesitan de las tecnologías inalámbricas, conformando la base estructural para su existencia. Estas tecnologías utilizan diferentes reglas y regulaciones para poder comunicar dispositivos y transmitir datos de forma segura y eficiente. A estas reglas se las llama protocolos y estándares de comunicación inalámbricas. Entre los más conocidos para el uso del IoT son el Bluetooth, Wi-fi, ZigBee, WiMAX, LoRa, y en algunos casos se utiliza la tecnología celular 5G. Cada una de ellas posee una regla específica para encriptar los datos, rige el rango de frecuencia a ser utilizado, su rango de cobertura, el tipo de multiplexación,

y entre otras características más, como se ve en la tabla 7, la cual muestra una comparación entre los diferentes tipos de tecnologías inalámbricas que pueden ser utilizadas en sistemas IoT.

TABLA VI. Comparación de las diferentes tecnologías inalámbricas utilizadas en el IoT [33, p. 10].

Característica de la tecnología.	**Bluetooth**	**LoRa**	**Wi-fi**	**WiMAX**	**ZigBee**
Autentificación	Llave de autentificación compartida	CCM	WPA2	CBC-MAC	CBC-MAC/Extensión de CCM
Duración de baterías	Bajo	Alto	Alto	Alto	Bajo
Protección de datos	16 bit CRC	128 bit CRC	32 bit CRC	128 bit CRC	16 bit CRC
Velocidad	1-24 Mbps	0.3-50 kbps	1 Mbps – 6.75 Gbps	1 Mbps – 1 Gbps	250 kbps
Consumo de energía	Medio	Muy bajo	Alto	Medio	Muy bajo
Banda de frecuencias	2.4 GHz	868/900 MHz	5-60 GHz	2-66 GHz	868/915 GHz
Propagación	FHSS	CSS (Espectro de dispersión Chirp)	DSSS, CCK, OFDM	OFDM	DSSS
Estándar	IEEE 802.15.1	IEEE 802.15.4 g	IEEE 802.11 a/c/b/d/g/n	IEEE 802.16	IEEE 802.15.4
Topología	Malla, estrella, árbol	Estrella de estrellas	Estrella, P2P	Red de acceso por radio, malla	Árbol, P2P, estrella y malla
Alcance	8-10 metros	< 30 km	20-100 metros	< 50 km	10-300 metros

Hardware compatible con el IoT

La tabla 8 presenta diferentes plataformas de hardware existentes que pueden operar con la tecnología IoT y que han atribuido a su desarrollo. Los parámetros por tomar en cuenta son la memoria, la tecnología inalámbrica soportada, puertos de entrada y salida, niveles de voltaje de operación, el tipo de procesador, entre otros.

TABLA VII. Tabla comparativa de varios hardware que soportan integración IoT [34, p. 294].

Parámetros	Arduino	D1 Wemos - NodeMCU	Intel Galileo Gen 2	Intel Edison	Raspberry-Pi B+	Electric Imp 003
Procesador	Atmega 328p-128p	Tensilica Xtensa LX106	Intel Quark SoC X1000	Intel Quark SoC X1000	Broadcom BCM2835 basado en SoC Núcleo de video ARM11 76JZF IV de 250 MHz	ARM Cortex M4F
Voltaje de operación	2.5 ~ 3.6 v	2.5 ~ 3.6 v	5 v	3.3 v	3.3 v	3.3 v
Velocidad de reloj	16 MHz -8 MHz	80 MHz	400 MHz	100 MHz	320 MHz	700 MHz
Ancho de banda del bus	8 bits	8 bits	32 bits	32 bits	32 bits	32 bits
Memoria del sistema	16 KB	16 KB	256 MB	1 GB	512 MB	120 KB
Memoria flash	4 MB	4 MB	8 MB	4 GB	-	4 MB
EEPROM	512 bytes	512 bytes	8 kb	-	-	-
Estándar de comunicación	IEEE 802.11 b/g/n, serial, bluetooth, Ethernet, serial.	IEEE 802.11 b/g/n, serial	IEEE 802.11 b/g/n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serial	IEEE 802.11 b/g/n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serial	IEEE 802.11 b/g/n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serial	IEEE 802.11 b/g/n, IEEE 802.15.4, 433RF, BLE 4.0, Ethernet, serial eléctric

						o Imp IDE
Ambiente de desarrollo	Arduino IDE	Arduino IDE, Lua NodeMCU	Arduino IDE	Arduino IDE, Eclipse, Intel XDK, C++	NOOBS	Squirrel
Puertos de entrada y salida	I2C, SPI, UART, I2S, GPIO	I2C, SPI, UART, I2S, GPIO	SPI, I2C, UART, GPIO	I2C, SPI, UART, I2S, GPIO	SPI, I2C, GPIO, UART, SDIO, CSI	SPI, I2C, UART, GPIO

Sistemas en la nube para el IoT

Una vez que el sistema IoT ha captado los datos, la nube IoT provee una rápida e idónea forma de almacenar esos datos, permitiendo su consulta y muestra posterior para su análisis. Existen tres tipos de servidores utilizados en el IoT según su funcionalidad, el servidor MQTT, HTTP y el servidor de almacenamiento. Gracias a las tecnologías de virtualización, el hardware puede ser aprovechado ya que estas son maquinas limitadas en lo visual, enfocándose en los recursos físicos como CPU, memoria, etcétera [35, p. 6].

- *Servidor HTTP:* este servidor está basado en el mecanismo de respuesta-petición de los usuarios que interactúan con él. Los usuarios pueden enviar peticiones de varios tipos como son el GET para obtener datos específicos del servidor, POST para enviar datos, PUT para actualizar datos, entre otros. Estas peticiones se envían por medio de una dirección web alojado en el servidor. Estas direcciones pueden alojar archivos de tipo HTML o JSON, los cuales pueden mostrar o modificar los datos según la aplicación [35, p. 6].
- *Servidor MQTT: e*l protocolo MQTT fue diseñado para tener una comunicación constante con el usuario y los dispositivos. Sin embargo, la latencia de los datos se puede reducir de alguna manera, pero este protocolo puede ser menos costoso ya que utiliza menor ancho de banda que el HTTP, siendo una gran ventaja cuando el volumen de datos es grande. De cierta forma este servidor permite una visualización en tiempo real de los datos [35, p. 6].

- *Servidor de almacenamiento:* este servidor es esencial ya que almacena los datos tomados por los sensores para su posterior análisis y toma de decisiones. Tradicionalmente se usan bases de datos comunes como lo son MySQL u Oracle, sin embargo, debido a las diferentes necesidades y aplicaciones del IoT estas han generados obstáculos para el rendimiento de la nube. Gracias a las bases de datos no relacionales, los sistemas IoT pueden almacenar datos de una forma más rápida [35, p. 6].

Aplicaciones de la nube en el IoT

Existen múltiples aplicaciones en lo que concierne a la nube, y el desenvolvimiento de cada una puede estar definida por sus dominios en el manejo de los datos como el manejo de dispositivos, visualización de datos, analíticas, entre otros. En la Fig. 6 se muestran algunas de las aplicaciones más comunes de la nube en el IoT.

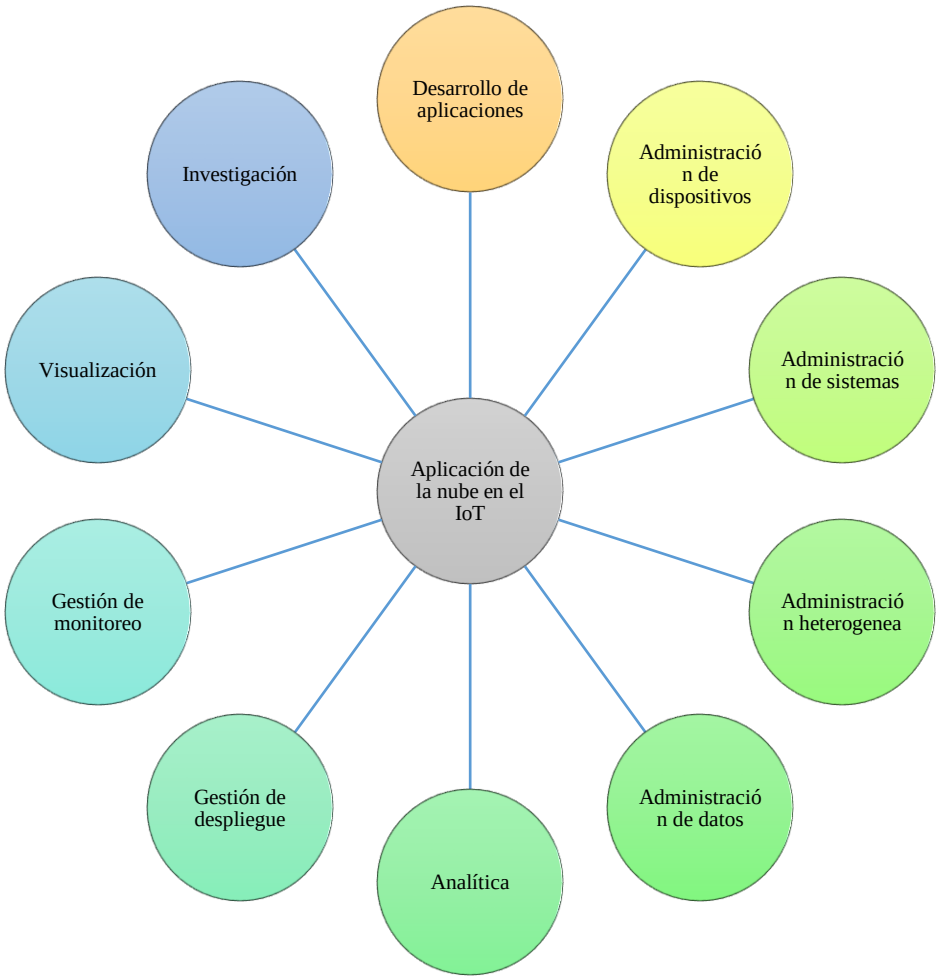


Fig. 6. Aplicaciones de la nube en el IoT [36, p. 37].

En el mercado existen múltiples opciones de servicios en la nube compatibles con IoT, pero por razones de tiempo se analizarán 14. En la tabla 9 se puede observar una comparación de los distintos servicios en la nube que trabajan con IoT, donde se analiza si estas pueden manejar almacenamiento de datos, visualización, manejo de dispositivos, investigación, desarrollo de aplicaciones, entre otras características más disponibles en cada servicio.

TABLA VIII. Dominios de los diferentes servicios en la nube [36, p. 38].

Plataforma IoT	**Desarrollo de aplicaciones**	**Administración de dispositivos**	**Administración de sistemas**	**Manejo heterogéneo**	**Manejo de datos**	**Analíticas**	**Gestión de despliegue**	**Administración de monitoreo**	**Visualización**	**Investigación**
Aer cloud			X			X		+		
Amazon	X	X			X	X		X	+	
Arkessa		X			+					
Axeda		X			+			X		
Carriots	+	X					+			
Etherios		+						X		
Exosite		X	+					X		
Nimbits					+	X				
Oracle IoT cloud			X	X	+		X		X	
Plotly						X	X		+	
SensorCloud		+				X		X		
Temboo	+									
Thethings.io	X		+					X		
Xively	X	+						X		

El signo “+” significa que los dominios de cada plataforma IoT son apropiados, que están destinados para esta aplicación de forma exclusiva. El signo “X” en cambio expresa que el dominio puede ser aplicable, aun si no es su campo de aplicación.

Software de gestión de la nube

Varias de los servicios en la nube ofrecen software de gestión de datos, pero en su mayoría son pagos y de gestión compleja. Los softwares de gestión en la nube permiten gestionar los datos que se almacenan en la nube, incluyendo un sinfín de servicios como el de manejo de correo electrónico, gestión de usuarios, sitio web, gestión de proyectos, entre otros. Con la ventaja de que son de código abierto y gratuitos en su mayoría de funcionalidades. En la tabla 10 se muestran varios softwares de gestión de la nube, de acuerdo con las funcionalidades que estos permiten, el código en el que se puede trabajar, bases de datos que maneja, campos de aplicación, entre otros aspectos.

TABLA IX. Comparación de los distintos softwares de gestión de la nube.

Elaborado por el autor.

Característica	**Odoo**	**ERPNext**	**Metasfresh**	**Dolibarr**	**Tryton**
Lenguajes	JavaScript, Python, XML	Python, JavaScript	Java, JavaScript, XML, SQL	PHP	Python
Aplicaciones	Gestión de proyectos, gestión de usuarios, contabilidad, gestión de datos, gestión de email, estudio clínico.	Fabricación, Servicios, educación, cuidado de la salud, agricultura.	Manejo de negocios, planificación de proyectos, compra, manejo de datos.	Servicios empresariales, gestión de datos, servicios de email.	Administración de compras, contabilidad, administración de datos, analítica de contabilidad.
Base de datos	PostgreSQL	MariaDB	PostgreSQL	MySQL	PostgreSQL
Genero	ERP, CRM	ERP, CRM	ERP, CRM, Contabilidad	ERP, CRM	ERP
Sistema operativo	Disponible en imagen virtual.	Disponible en imagen virtual.	Linux	Windows, MAC OS X, Linux.	Windows, MAC OS X, Linux.

1.1.1.4. Telemedicina

Introducción

El avance de las tecnologías de la comunicación ha impulsado a los sistemas de salud a implementar sistemas de intercambio de información médica. Inicialmente adaptado para aquellas personas que se encuentran geográficamente imposibilitadas para tener acceso a sistemas de salud pública. Gracias a esto, estas personas tienen acceso a nuevos métodos de atención médica. Con el constante impulso de las tecnologías de la comunicación, estas han permitido ir más lejos que el objetivo inicial de la telemedicina, facultando a los galenos ayudar a colegas a distancia en toma de decisiones, además de brindar formaciones y capacitaciones. Estas características facultan a los sistemas de salud a compartir información cuando la distancia es un problema, mejorando la atención a los pacientes y el ambiente laboral del personal sanitario [37, pp. 840-841].

Definición

La telemedicina se define simplemente como la atención médica a distancia. Esta ocurre gracias a la utilización de las tecnologías de la comunicación y de la información para el intercambio de datos vitales para el diagnóstico o prevención de enfermedades, entre otras aplicaciones. Esta puede ser tan simple como que dos profesionales de la salud están hablando de algún caso medico mediante intercambio de mansajes de texto por alguna aplicación o mail, hasta la utilización de tecnología avanzada para realizar consultas, intervenciones médicas, cirugías, prescripción de medicamentos, diagnósticos, o incluso la capacitación y formación profesional. Todas estas características permiten reducir costos de operación, optimizar tiempos de los procesos médicos y permitiendo el acceso a los servicios de salud a los pacientes donde la geografía es complicada, o están lejos de los servicios de salud [37, p. 841].

Aplicaciones de la telemedicina

Las aplicaciones de la telemedicina pueden variar según el caso médico que el profesional de la salud este atendiendo, pero estas pueden cubrir la mayor parte de áreas de la salud.

- *Tele-prevención:* esta trata de prever que el paciente sufra algún daño en su salud, además de informar de la existencia de algún riesgo, utilizando las TIC (Tecnologías de la información y comunicación) [37, p. 841].
- *Tele-diagnóstico:* ayuda a los galenos en la toma de decisiones en casos médicos, ya sea por teleconferencia o mediante el uso de bases de datos [37, p. 842].
- *Tele-consulta:* en este caso el paciente se pone en contacto con un especialista de acuerdo con su cuadro médico para recibir prescripciones o tratamientos. También puede ser utilizado para el intercambio de información entre el personal técnico y un galeno, o entre dos especialistas. Esta puede ser de tipo Tele-junta médica, donde médicos de dos o más instituciones médicas se ponen en contacto para el intercambio de información para un caso en particular. También puede ser de tipo Telepresencia, donde un médico se pone en contacto con un paciente, permitiéndole discutir síntomas, prescripciones o tratamientos con otro médico, asistiéndolo a distancia [37, p. 842].
- *Tele-emergencias:* se utilizan equipos médicos para realizar diagnósticos, monitoreando signos vitales con la ayuda de profesionales de la salud. Esta información es compartida con el medico a distancia para la toma de decisiones necesarias en cada caso [37, p. 842].
- *Tele-monitorización:* se basa en la toma de datos vitales del paciente para el diagnóstico médico, estos datos pueden ser radiografías, signos vitales, electrocardiogramas, etc. Esta se utiliza en su mayor parte en unidades de cuidados intensivos cuando existen riesgos biológicos o en pacientes con enfermedades crónicas que no pueden salir de su domicilio [37, p. 842].

Un sistema de telemedicina funciona gracias a la ayuda de las comunicaciones y de la transmisión de la información por este medio, siendo el más común internet. Desde el usuario o paciente la información es tomada por diferentes métodos, después esta es enviada por el internet hasta el medico que brinda una segunda opinión o un diagnóstico de algún caso médico a un colega o a un paciente de forma directa. Esta estructura general se puede apreciar en la Fig. 7 [37, p. 841].

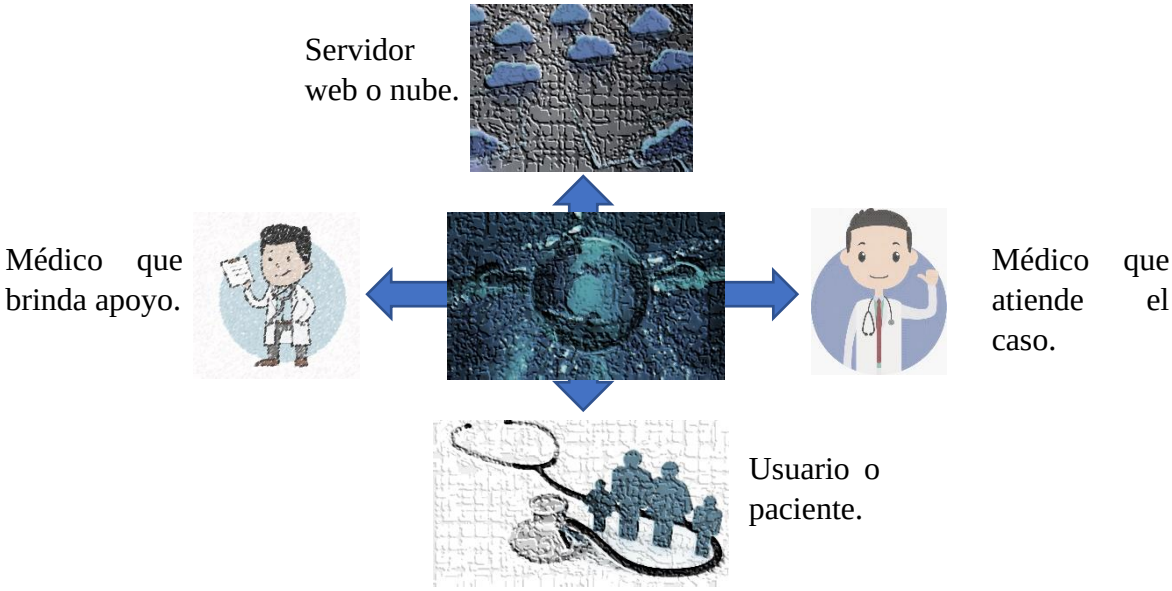


Fig. 7. Estructura general de un sistema de telemedicina [37, p. 841].

1.1.1.5. Rol del IoT en la telemedicina

Introducción

La implementación del internet de las cosas en la telemedicina ha recibido una considerable atención en los últimos años debido al increíble avance de las tecnologías de la información y comunicación, y el acceso a dispositivos inteligentes como los smartphones, tabletas, entre otros [38]. Estos dispositivos en conjunto con el internet pueden brindar una intervención rápida y oportuna en diversos casos médicos. La aplicación del internet de las cosas en la telemedicina puede ayudar a los médicos en la atención a pacientes que necesitan un monitoreo constante de su estado, y que no se encuentran en la misma ubicación geográfica [39, p. 1]. La tabla 11 muestra diferentes aplicaciones del IoT en la telemedicina, detallando la arquitectura que puede ser usada. Muchas de las aplicaciones más conocidas del internet de las cosas están vinculadas con el control de pacientes con enfermedades crónicas, entrenamiento físico, sistemas de monitoreo de la salud y cuidado ambulatorio de las personas ancianas [33, p. 1].

TABLA X. Internet de las cosas y su aplicación en la telemedicina [39, p. 2]

Aplicación	Arquitectura
Monitorización de la salud de pacientes ancianos.	Proveedor de almacenamiento en la nube, sensor de ECG (Electrocardiograma), aplicaciones móviles.
Imagenología médica.	Nube, dispositivos para imagenología médica.
Monitoreo de ECG.	Sistema basado en la nube, sensor de ECG, aplicación para teléfono móvil, software de sobremesa.
Monitoreo de anestesia.	Monitorización terapéutica del fármaco, aplicación para dispositivo móvil, reloj inteligente.

Salud humana y el Internet de las cosas

El objetivo principal del internet de las cosas es brindar acceso y control a un sin número de cosas conectadas al internet. La red que provee los datos forma un rol principal para la conexión de los dispositivos de cuidado médico. Como se muestra en la Fig. 8, está constituida por el tipo de topología, la arquitectura y la plataforma a ser utilizada, brindando los requerimientos estructurales para su correcto funcionamiento [33, p. 2].

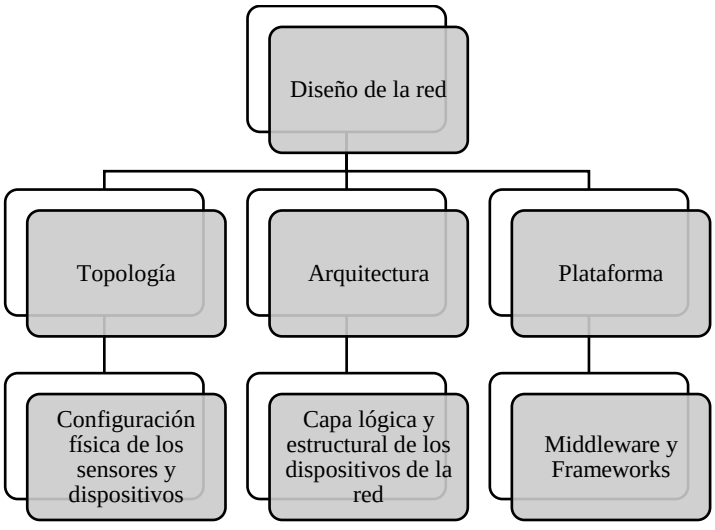


Fig. 8. Diseño de la red IoT para la salud humana [33, p. 2].

- *Topología:* la topología consiste en el despliegue físico de los elementos necesarios para el cuidado de la salud, como son los sensores y puertas de

enlaces desde una perspectiva de comunicación. Los factores que influyen en la elección de una topología son el costo, consumo de energía, tecnologías de comunicación y confiabilidad. Todos estos factores están influenciados por la características, posibilidades y rendimiento de las topologías, como lo son la latencia, tolerancia a fallos, escalabilidad de los sensores, y los saltos de la información antes de llegar a la puerta de enlace más cercana [33, pp. 2-3].

- *Arquitectura:* la arquitectura es como un esquema de sistema físico, virtual o hibrido, los cuales están compuestos por los dispositivos físicos, sensores, actuadores, protocolos específicos del usuario, plataformas en la nube, capas de comunicación, organización funcional y sus principios de trabajo [33, p. 4].
- *Plataforma:* esta ofrece la conexión y la plataforma que conecta a los dispositivos IoT con la nube. Convencionalmente estos pueden administrar, controlar, monitorear y establecer una conexión segura entre todos los dispositivos conectados. Uno de los retos es diseñar una plataforma que brinda interoperabilidad entre todos los dispositivos involucrados [33, p. 5].

En la Fig. 9 se puede observar un modelo del sistema IoT para la salud humana en el hogar, este permite la integración de distintos dispositivos en función de los protocolos, softwares y topología con los que estos funcionan.

- *Cosas médicas:* esta capa esta encargada de captar los datos médicos del paciente por medio de sensores, está compuesta por dispositivos Weareables inteligentes, sensores aplicados en partes específicas del cuerpo del paciente, y las aplicaciones móviles que pueden estar alojadas en smartphones, tabletas, relojes inteligentes, entre otros [33, p. 6].
- *Conectividad:* esta capa se encarga de llevar los datos captados por las cosas medicas hacia la capa de servicios de administración de los datos. Esta enruta los protocolos de las dos capas, permitiendo la interconectividad, la seguridad y la versatilidad del sistema de telemedicina [33, p. 7].
- *Servicio de administración:* esta compone la parte central del sistema, donde se ejecutan los procesos de almacenamiento, procesamiento, administración de los dispositivos conectados al sistema, además de la actuación y control de estos. Esta capa además maneja los protocolos de comunicación que la capa de

conectividad para llevar a cabo la comunicación y las funciones de nube [33, p. 7].

- *Interfaz de usuario y análisis de datos:* en esta capa se realizan todos los procesos de analítica de datos para el cuidado de la salud. Además, estos proveen una visualización de los datos médicos captados en la primera capa, para realizar el diagnostico, enviar prescripciones, y la toma de decisiones en general [33, p. 7].

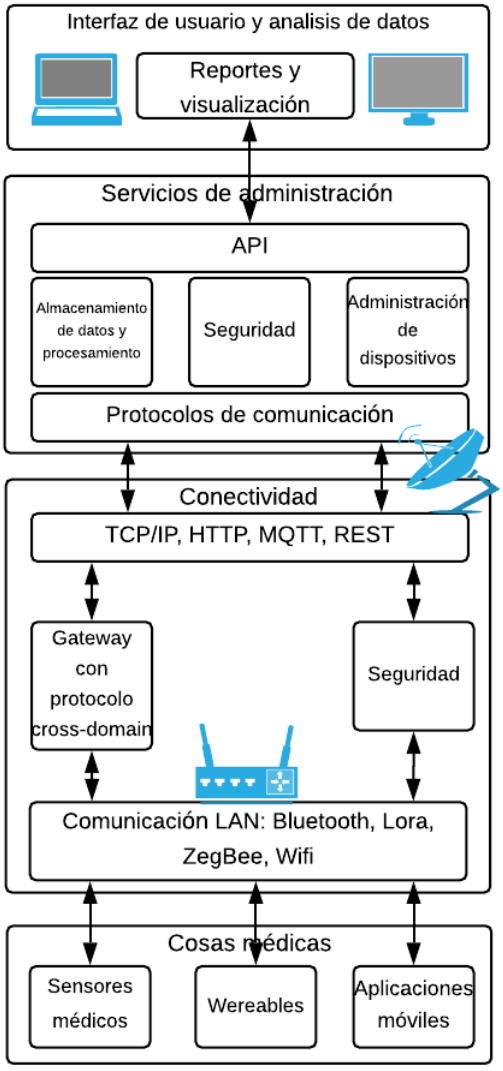


Fig. 9. Sistema de telemedicina en el hogar basado en IoT [33, p. 9].

Sensores médicos Weareables

Dispositivos de sensado han permitido obtener datos vitales del paciente, entre los cuales se pueden encontrar electrocardiogramas, presión sanguínea, temperatura

corporal, pulso oximetría, niveles de glucosa en la sangre, entre otros. Debido a las limitaciones de hardware, especialmente en dispositivos Weareables, todas estas variables son difíciles de manipular y procesar, y los más común es la implementación de uno o dos variables al mismo tiempo. Con los recientes avances en la computación en la nube, dichos dispositivos son capaces de procesar esos datos y subirlos a la nube por medio de protocolos de comunicación inalámbricos [40, p. 3]. Muchos de estos sensores trabajan con tecnologías ópticas, los cuales poseen alta sensibilidad y la capacidad de medir varios parámetros, además de permitir comunicaciones de tipo I2C o serial, o en algunos casos son de tipo analógicos.

Los dispositivos Weareables son capaces de medir parámetros biomédicos del paciente en espacios reducidos, y pueden medir tres categorías de signos vitales, como son los sensores de ritmo cardiaco, sensores de temperatura corporal y sensores de presión sanguínea.

- *Sensores de ritmo cardiaco:* estos sensores son capaces de captar con que regularidad se ejecuta un latido del corazón, que de echo refleja la actividad del miocardio. Estos sensores son utilizados para el monitoreo de pacientes con problemas cardiovasculares. Su función es la de permitir el estudio del comportamiento de las funciones del corazón a través de un dispositivo colocado en el lóbulo de la oreja, dedos, o brazo del paciente. Varios dispositivos pueden trabajan con tecnologías ópticas para detectar cambios continuos en la sangre, además de detectar variaciones en el color, lo cual permite medir la oxigenación [40, p. 4].
- *Sensores de temperatura:* estos permiten capturar la temperatura en el ambiente o corporal y mostrarlos en grados Celsius. Uno de los más populares es el sensor analógico LM35, que utiliza una alimentación de 5 voltios [40, pp. 4-5].
- *Sensores de presión sanguínea:* estos sensores son capaces de medir la presión sanguínea mediante una técnica llamada oscilométrica, la cual mide la presión sistólica, diastólica y la presión media. Esta característica corporal varía dependiendo de muchos factores, como la posición corporal, estado emocional, ejercicio físico o en las diferentes etapas del sueño [40, p. 5].

En la tabla 12 se presenta una comparación de todos los sensores Weareables según su aplicación, funcionamiento, voltajes, consumo de corriente entre otros parámetros.

TABLA XI. Tabla de comparación de sensores médicos Weareables.

Elaborado por el investigador.

Característica	**MAX30100**	**Sensor de pulso Amped**	**Modulo AD8232**	**LM35**	**Modulo KY-039**
Aplicaciones	Weareables, dispositivos de asistencia física, dispositivos de monitorización médica.	Weareables, dispositivos de asistencia física, dispositivos de monitorización médica.	Monitorización del corazón	Medición de temperatura ambiental y corporal en grados centígrados.	Medición del ritmo cardiaco en dispositivos de medición médica
Consumo de corriente	20 mA	4 mA	5 mA	60 mA	4 mA
Voltaje de alimentación	1.8 – 5 v	3 – 5v	3.3 v	4 – 30 v	3 -5 v
Comunicación	I2C	-	-	-	-
Dimensiones	5.6mm x 2.8mm x 1.2mm	1,6 cm x 1,6 cm x 0,5 cm	3.5 cm x 3 cm	4.30 mm x 4.30 mm	25 x 12 x 12 mm
Tecnología	Óptica	Óptica	Electrodos biomédicos	-	Óptica
Precio ($)	8	6	25	2	3.50

1.2. Objetivos

1.2.1. Objetivo General

Implementar un sistema de telemedicina basado en IoT en un ambiente de Smart TV.

1.2.2. Objetivos Específicos

- Establecer las tecnologías más viables a utilizar en el diseño del sistema de telemedicina en un ambiente de Smart TV.

Aquí se realizará una investigación para conocer las tecnologías que son compatibles tanto para un ambiente Smart TV como para el IoT, las cuales permitirán el diseño estructural del sistema.

Para cumplir este objetivo se deben desarrollar las siguientes actividades:

1. Investigación de las tecnologías utilizadas en la telemedicina.
2. Análisis de la estructura de un sistema IoT para la adquisición de datos de signos vitales.
3. Análisis de los sensores que se utilizan para la obtención de datos de signos vitales.
4. Identificación de los recursos tecnológicos de red que son necesarios para el sistema IoT.

- Diseñar un sistema para el procesamiento y presentación de los signos vitales en un ambiente Smart TV.

Para esta sección del proyecto se comienza con el diseño del sistema de telemedicina, utilizando las tecnologías investigadas en el objetivo anterior.

Para cumplir este objetivo, se deben cumplir las siguientes actividades:

1. Selección de los dispositivos electrónicos necesarios para la implementación del sistema IoT.
2. Elaboración de un circuito electrónico de toma de signos vitales basado en IoT.
3. Vinculación de los datos obtenidos de las señales vitales con el ambiente Smart TV.

- Elaborar una interfaz para la presentación de los signos vitales en un ambiente Smart TV basado en IoT.

Para finalizar el sistema de telemedicina se diseña la interfaz que interactúa con los datos recogidos del paciente y que envía peticiones a la nube.

A fin de cumplir con este objetivo es necesario cumplir con las siguientes actividades:

1. Diseño del ambiente Smart TV para procesamiento y exhibición de los signos vitales.
2. Ejecución de pruebas de un sistema de telemedicina basado en IoT en un ambiente de Smart TV.

3. Análisis de resultados de un sistema de telemedicina basado en IoT en un ambiente de Smart TV.

CAPÍTULO II

METODOLOGÍA

2.1. Materiales

2.1.1. Selección de software y hardware.

2.1.1.1. Capa de sensado.

Sensado de signos vitales

El circuito de sensado de signos vitales está compuesto de dos dispositivos: el dispositivo de sensado y el dispositivo de procesamiento, comunicación y muestra de datos. Para el dispositivo de sensado se propone utilizar el sensor MAX30100, que además de brindar datos de pulsaciones por minuto del corazón y oxigenación en la sangre, este puedo comunicarse por medio del protocolo I2C. Además de que su tamaño es reducido y puede ser implementado en una pulsera Weareable. Para el dispositivo de procesamiento y comunicación se propone utilizar la placa electrónica Arduino Mini Pro en su versión de 5V a 16MHz de oscilación, que además de manejar el protocolo de comunicación bluetooth utilizando el módulo HC-05 compatible con el IoT, posee un puerto de comunicación I2C para la obtención de los datos del sensor y la muestra de estos en una pantalla OLED de 128x64 pixeles, un puerto analógico para la obtención del nivel de batería disponible, y puertos digitales para la interrupción de las lecturas del sensor MAX30100 y lectura del estado de conexión del módulo HC-05. Por demás, esta placa tiene una dimensión de 33.02 mm por 17.78 mm, siendo una ventaja para disminuir el tamaño de la pulsera Weareable. Se incluye también en circuito de carga para la batería dentro de la pulsera para evitar que cuando esta se descargue, se deba desmontar la pulsera y realizar la carga, proporcionando una medición continua de los datos. En la Fig. 10 se aprecia el diagrama de funcionamiento del sensado de signos vitales.

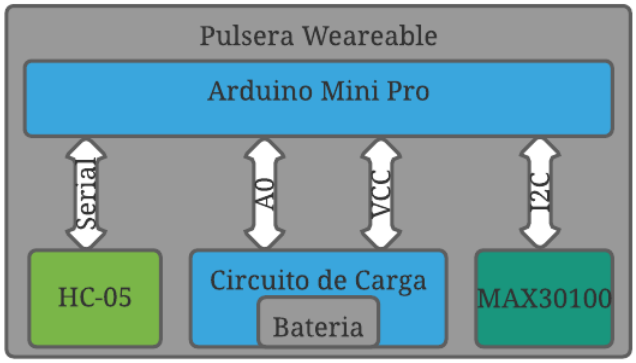


Fig. 10. Proceso de sensado de signos vitales.
Elaborado por el investigador.

2.1.1.2. Capa de servicios de administración en la nube.

Alojamiento en la nube

Para el alojamiento en la nube se propone alojar los datos y el software de administración en la nube que proporciona Amazon con sus servicios llamados AWS (Servicios Web de Amazon), que proporcionar 30 GB de almacenamiento y que son gratuitos durante 12 meses. Además, este servicio de nube posee aplicabilidad con el IoT, ya que posee varios servicios de administración de datos en este campo. El servicio gratuito de Amazon también permite seleccionar varios tipos de sistemas operativos virtuales como Ubuntu Server, para la gestión de bases de datos y de ambientes virtuales, en donde son alojados los softwares de administración de datos. También brinda un servicio de DNS y de protocolos de comunicación TCP/IP y HTTP que serán necesario para el funcionamiento del sistema de telemedicina. Por demás, se propone este servicio ya que el investigador posee experiencia en su manejo, tanto del sistema operativo Ubuntu Server y del panel de administración de AWS. Este servicio gratuito de Amazon llamado EC2 (Servidor virtual) de tipo “t2.micro” tiene las siguientes características técnicas:

- Ubuntu server 18.04.
- Estructura amd64 (x86).
- 1 GB de RAM.
- Manejo por consola.
- 30 GB de espacio en disco SSD (Disco de Estado Sólido).

- Una interfaz de red eth0.
- DNS IPV4 fijo público.
- SSH por medio de claves privadas.

Software de administración

El software de administración permitirá el desarrollo rápido y eficiente del sistema de telemedicina ya que este proporciona una interfaz web en la cual se pueden observar los datos de la base de datos de forma organizada. Además de que brinda soporte para la comunicación e interacción de los diferentes elementos del sistema mediante APIs, programadas en Python. Este software suele utilizar de 300MBytes a 500 MBytes de memoria RAM para su correcto funcionamiento.

En la Fig. 11 se muestra un esquema de funcionamiento de la capa de servicios de administración de datos.

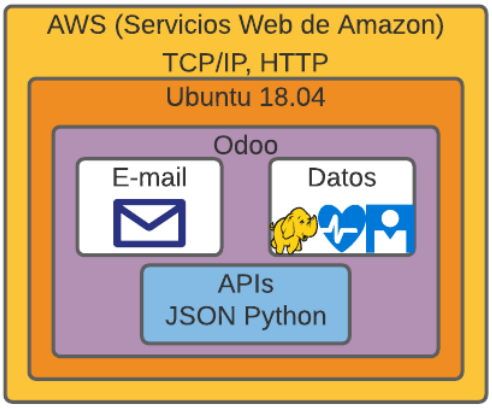


Fig. 11. Capa de servicios de administración en la nube.
Realizado por el investigador.

2.1.1.3. Ambiente Smart TV, aplicación IoT

Se propone implementar el ambiente Smart TV en el sistema operativo Android TV, el cual según se analizó en el capítulo 1, posee mayor cobertura de mercado a nivel mundial en lo que son dispositivos de televisión inteligente, y aún más en los llamados tv-box. La interfaz de usuario se desarrollará en el framework de programación React-Native, el cual posee soporte para los televisores Android TV, y que además posee la característica de desarrollo multiplataforma, permitiendo la mayor cobertura en lo que

se trata el desarrollo para ambientes Smart TV. Este crea una aplicación nativa, la cual permitirá a la aplicación utilizar las características de hardware del dispositivo tv-box para poder comunicarse con la nube, evitando latencias y ejecución de varias páginas de navegación e interacción con el usuario sin problemas de rendimiento. Para el ambiente Smart TV se propone utilizar un tv-box de la marca SpeedBox fusión, ya que posee conexión de bluetooth nativa y un precio bajo a comparación con un televisor con el sistema operativo incluido, además trabaja con comunicación inalámbrica IEEE 802.11 compatible con el IoT y el protocolo bluetooth IEEE 802.15.1. En la Fig. 12 se muestra un diagrama de funcionamiento del ambiente Smart TV.

Fig. 12. Funcionamiento del ambiente Smart TV.

Elaborado por el investigador.

2.2. Métodos

2.2.1. Modalidad de investigación

Este proyecto es de tipo investigativo y de desarrollo o I+D ya que se realizó una investigación de sustento, la cual se utilizó en el desarrollo de este proyecto. Las investigaciones se desarrollaron en bases de datos y bibliotecas virtuales proporcionado por el sistema integrado de la universidad. Las cuales proporcionan revistas, libros, artículos y conferencias de carácter científicos, relacionados con el tema de este proyecto para poder sustentarlo de manera técnica e ingenieril.

2.2.2. Recolección de información

Se utilizaron las bibliotecas virtuales y bases de datos libres proporcionados por el sistema integrado de la universidad, enfocándose en las temáticas que se exponen en este proyecto. Además, la información brindada por el tutor fue vital para el desarrollo de esta investigación.

2.2.3. Procesamiento y análisis de datos

El procesamiento y análisis de datos se llevó a cabo gracias al seguimiento de los siguientes pasos:

- Priorizar la indagación de información.
  - Definir qué información buscar.
  - Definir métodos de búsqueda de información.
- Establecer medios de almacenamiento de la información, para su organización y respaldo.
- Comprender la información obtenida mediante métodos de lectura comprensiva.
- Entender los resultados obtenidos.

2.2.4. Desarrollo del proyecto

- Investigación de las tecnologías utilizadas en la telemedicina.
- Análisis de la estructura de un sistema IoT para la adquisición de datos de signos vitales.
- Análisis de los sensores que se utilizan para la obtención de datos de signos vitales.
- Identificación de los recursos tecnológicos de red que son necesarios para el sistema IoT.
- Selección de los dispositivos electrónicos necesarios para la implementación del sistema IoT.
- Elaboración de un circuito electrónico de toma de signos vitales basado en IoT.

- Vinculación de los datos obtenidos de las señales vitales con el ambiente Smart TV.
- Diseño del ambiente Smart TV para procesamiento y exhibición de los signos vitales.
- Ejecución de pruebas de un sistema de telemedicina basado en IoT en un ambiente de Smart TV.
- Análisis de resultados de un sistema de telemedicina basado en IoT en un ambiente de Smart TV.

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

3.1. Análisis y discusión de resultados

3.1.1. Desarrollo de la propuesta

En este proyecto se implementa un sistema de telemedicina en un ambiente Smart TV con el monitoreo de signos vitales. La arquitectura del prototipo de sistema se muestra en la Fig. 13, en función de la arquitectura de un sistema de telemedicina IoT en el hogar, analizado en la sección 1.1.1.5.

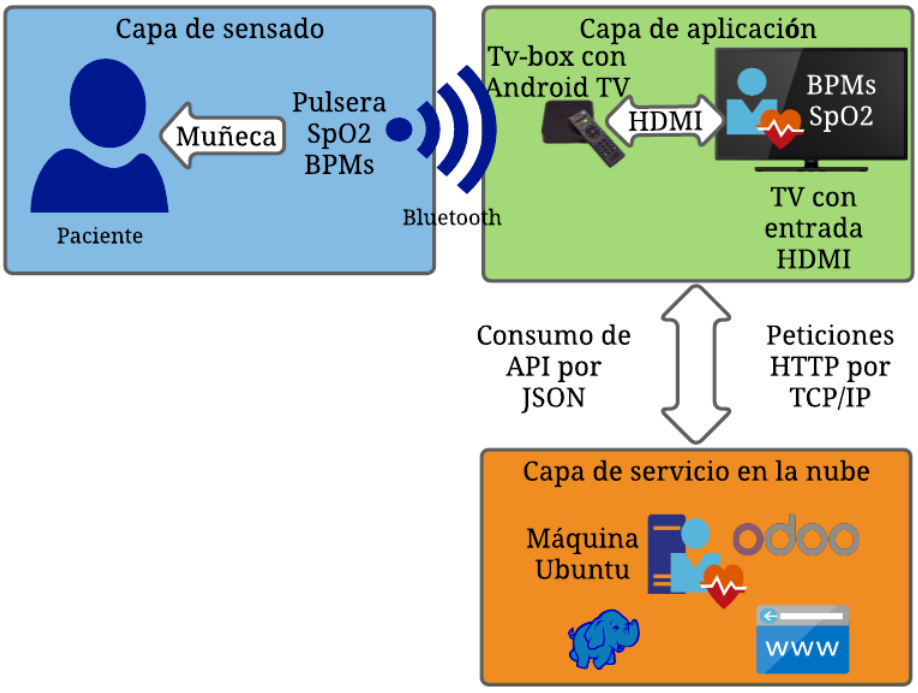


Fig. 13. Arquitectura del Sistema de Telemedicina con monitoreo de signos vitales basado en IoT.

Elaborado por el investigador.

Capa de sensado

Para el sensado de los signos vitales se diseñó un brazalete Weareable que contiene el sensor y el dispositivo de procesamiento y de comunicación para enviar los datos a la aplicación por medio de bluetooth. El sensor por utilizar es el MAX30100, que trabaja a un voltaje de 3.3 voltios de alimentación, y que posee un puerto de comunicación I2C. El dispositivo de procesamiento y comunicación es el Arduino Pro Mini 5V a

16Mhz de oscilación, que trabaja con el módulo bluetooth HC-05, el cual maneja el protocolo IEEE 802.15.1. Este circuito podrá interactuar con la aplicación de Android TV al enviar los datos sensados desde el sensor MAX30100 y mostrándolos, al igual que el nombre del usuario seleccionado, mediante una pantalla OLED de 128x64 pixeles que posee un puerto de comunicación I2C con 3.3 voltios de alimentación.

Capa de aplicación

En esta capa se implementa el ambiente Smart TV para poder visualizar y monitorear los signos vitales del paciente, los cuales son las pulsaciones por minuto (BPM) y el nivel de oxígeno en la sangre (SpO2). Esta aplicación provee la interfaz gráfica para la interacción con el paciente, mostrando los nombres de los pacientes que han sido creados, además de un botón que permite crear nuevos usuarios. Cada usuario tendrá la opción de agregar su nombre y edad. Los datos de BPMs y SpO2 serán recibidos por medio de bluetooth desde la pulsera Weareable para ser almacenados en la nube utilizando peticiones HTTP hechas por la aplicación de Android TV al servicio en la nube. Cada usuario tendrá un gráfico donde se mostrarán los datos en forma de historial, para que el usuario pueda visualizar el avance de su salud. Esta aplicación permitirá también acceder a todas las prescripciones enviadas desde Odoo, permitiendo al usuario acceder a ellas utilizando filtros de fechas, al igual que el gráfico.

Capa de servicios en la nube AWS

Esta capa esta encargada de almacenar y administrar los datos que son enviados desde la capa de sensado y la capa de aplicación, a través del software de administración Odoo, alojado en el servicio de Amazon. Odoo facilita el acceso a los datos mediante una interfaz web, permitiendo al médico encargado enviar prescripciones cuando es notificado por el sistema, estas notificaciones se mostrarán cuando los signos vitales del usuario sean anormales. Además, brinda servicios web para la consulta y actualización de datos, mediante la interacción con APIs consumibles para la capa de aplicación mediante el protocolo HTTP.

3.1.1.2.Direccionamiento y topología

Para el funcionamiento del sistema de telemedicina serán necesarias dos redes distintas: red inalámbrica local para la conexión de la capa de aplicación y otra para la conexión con el servicio en la nube de Amazon. Cabe mencionar también una conexión I2C para el sensor y pantalla del brazalete Weareable.

Direccionamiento I2C

La pantalla OLED y el sensor MAX30100 necesitan dos direcciones I2C diferentes para su correcto funcionamiento, estas serán definidas mediante software en el IDE de Arduino. Las direcciones de los dispositivos se muestran en la tabla 12.

TABLA XII. Direcciones I2C de los dispositivos del brazalete Weareable.
Elaborado por el investigador.

Dispositivo	**Dirección I2C hexadecimal de 8 bits**
MAX30100	0x57
OLED 128x64	0x3C

Direccionamiento de la red de área local inalámbrica

En esta red estará conectado el dispositivo tv-box. El router HG110 brindará conexión a internet mediante el protocolo de configuración dinámica de host (DHCP), como se puede observar en la tabla 13. Estos dispositivos se comunicarán con los protocolos IEEE 820.11 b/g/n, en la red 192.168.1.0/24, ya que varios dispositivos del hogar obtienen acceso a internet mediante esta red, se la define de acuerdo con su configuración de fábrica.

TABLA XIII. Direccionamiento de la red inalámbrica local.

Realizado por el investigador.

Dispositivo	**Dirección**	**Duración en segundos**	**Rango**	**Mascara**
Router de conexión a internet HG110	192.168.1.1	-	-	255.255.255.0
Tv-box	Definida por DHCP	86400	192.168.1.2 – 192.168.1.62	255.255.255.0

Direccionamiento de red para la capa de servicio web

Amazon provee una dirección IPv4 o DNS fijos, por lo que el usuario puede acceder desde cualquier lugar del planeta mediante el protocolo TCP/IP en su versión 4. Para esta investigación se realizarán las pruebas de funcionamiento desde la ubicación del investigador, cuyas direcciones se definen en la tabla 14. La dirección IP de la capa de conectividad puede cambiar ya que el IP que provisiona la empresa CNT es dinámica.

TABLA XIV. Direccionamiento de red para la capa de servicios web.

Elaborado por el investigador.

Capa	**Dirección IPv4 publica**	**DNS publico**
Capa de servicio web Amazon	18.222.17.116/8	ec2-18-222-17-116.us-east-2.compute.amazonaws.com
IP pública usuario	190.152.129.77/16	-

Topología

La estructura de la topología con sus respectivas direcciones físicas se muestra en la Fig. 14.

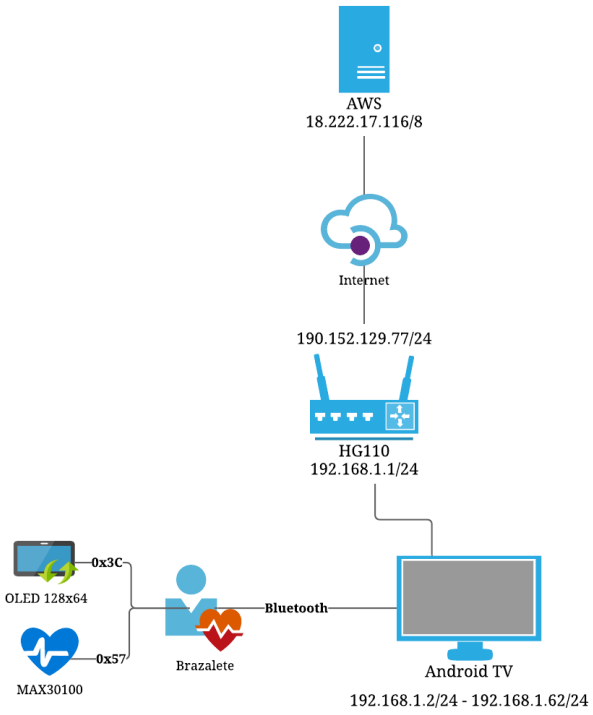


Fig. 14. Topología de red del sistema de telemedicina.
Elaborado por el investigador.

3.1.1.3. Implementación del servicio en la nube.

3.1.1.3.1. Programación del módulo de Odoo

Para comenzar con la programación del módulo es necesario tener instalada tanto la instancia en el servicio de AWS como el software Odoo, para lo cual se puede revisar el ANEXO A y B para instalar y levantar estos servicios. Las funcionalidades de Odoo se ejecutan a partir de la creación de módulos que están escritos en lenguaje Python y XML. Para la implementación de estas funcionalidades es necesario crear un módulo instalable, dentro de una carpeta en la dirección "*/odoo/custom/custom-addons*".

En esta dirección se ingresa el comando "*scaffold"* seguido del nombre del módulo, que en este caso es "*tesis*". El comando se muestra en la Fig. 15.

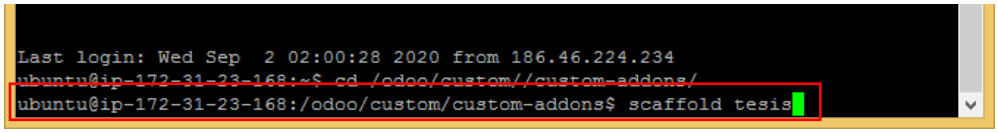


Fig. 15. Creación del módulo "tesis".

Elaborado por el investigador.

Una vez creado el módulo este debe ser modificado para que brinde las funcionalidades necesarias del sistema. Esto se logra al estructurar el módulo como se muestra en la Fig. 16.

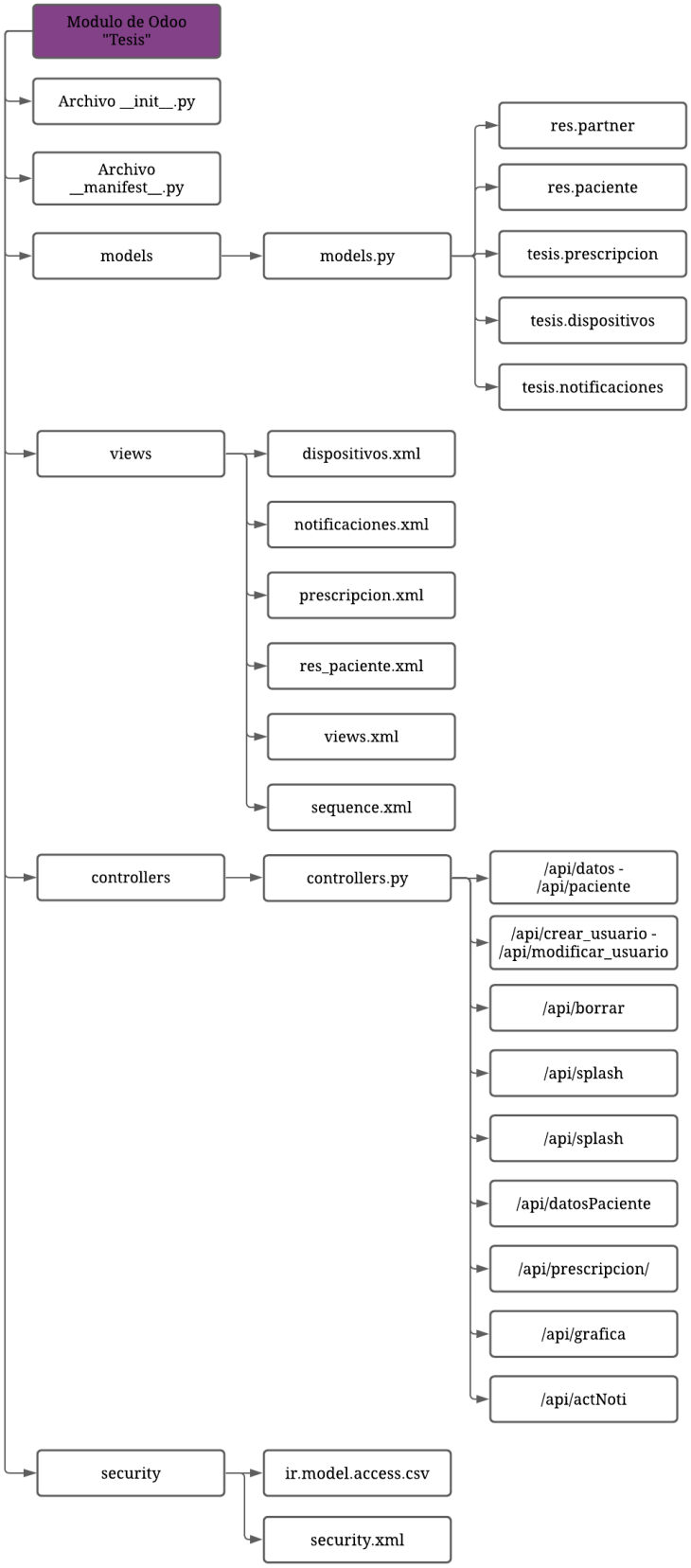


Fig. 16. Estructura de archivos del módulo de Odoo.

Elaborado por el investigador.

__init__.py

En este archivo se importan todos los archivos escritos en Python que el módulo va a utilizar, los cuales son el "*controllers*", "*models*" y "*static*".

```
from . import controllers
from . import models
from . import static
```

__manifest__.py

Este es el directorio raíz del módulo donde se define el nombre, descripción y los archivos "data" los cuales son ejecutados cada vez que el módulo es instalado. Esta sección contiene todos los archivos de configuración, vistas, metadatos entre otros. Estos se definen como se ve a continuación.

```
'data': [
    'security/ir.model.access.csv',
    'security/security.xml',
    'views/sequence.xml',
    'views/views.xml',
    'views/res_paciente.xml',
    'views/prescripcion.xml',
    'views/dispositivos.xml',
    'views/notificaciones.xml'
],
```

Models

En esta carpeta se definen los objetos de negocio en forma de clases con los que Odoo trabaja. Estos en conjunto con los archivos de "*data*" evitan escribir código de peticiones SQL para realizar las consultas a la base de datos. Estos permiten interactuar con las vistas en Odoo al pasarle datos esenciales y para controlar la información que se muestra de la base de datos.

- **class ResPartner:** esta clase hereda todas las "*props*" de "*res.partner*" que son todos los usuarios creados en Odoo que pueden ser Usuarios, Empleados, etc. En este caso, se establece una nueva categoría "pacientes" al agregar un booleano dentro de la clase. Además, se le asocia a un dispositivo y se le crea un campo edad de tipo entero.

- **class ResPaciente:** esta clase permite almacenar a todos los usuarios que estén dentro de la categoría “paciente”, crea dos nuevos campos que son los signos vitales y relaciona al usuario con la clase anterior con un campo de tipo “Many2one”.
- **class Prescripción:** este va a crear varios campos los cuales van a interactuar con el usuario en las diferentes vistas, como son el mensaje de tipo texto, el estado (enviado o borrador), si el mensaje ha sido leído por el usuario de la aplicación, entre otros. También, en esta se define la dirección web del socket a utilizar para el envío de las notificaciones y se le pasa un objeto llamado “data” el cual almacena diferentes parámetros que serán leídos desde la aplicación, para después mostrarla al usuario en forma de notificación.

```
data = {
    'tipo': 'notificacion',
    'titulo': 'Prescripción',
    'mensaje': 'Tienes una nueva prescripción médica',
    'prescripcion': self.id,
    'paciente': self.res_paciente.name,
    'dispositivo': self.res_paciente.dispositivo.codigo
}
```

- **class Notificaciones:** en esta se definen distintos campos propios de cada notificación, como son su nombre, campo atendido, los signos vitales que han disparado la aplicación entre otros. En este se define también una función que pasa la información a la clase “*class Prescripcion*” para mostrar el formulario de envío ya que, al responderla se creará y enviará una prescripción.

En la Fig. 17 se muestra un diagrama UML donde se detallan los distintos usuarios y modelos con sus atributos y funciones que Odoo utiliza para las vistas. El código completo del archivo “Models” se encuentra en el ANEXO C.

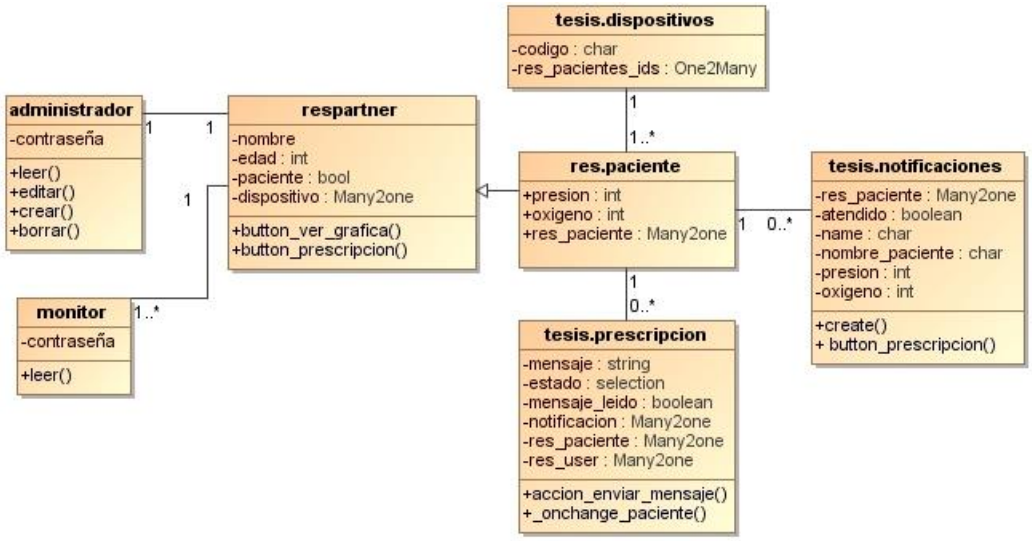


Fig. 17. Diagrama UML de los usuarios y modelos que son parte del proyecto.

Elaborado por el investigador.

Vistas en Odoo

Una vez definidos los modelos, es necesario mostrar los datos que se han definido utilizando los archivos dentro de la carpeta "*views*". Estos archivos consultan los datos desde el lado del cliente, llamando al modelo al cual son asociados y presenta la información alojada en la base de datos del servidor. Este proceso se muestra en la Fig. 18.

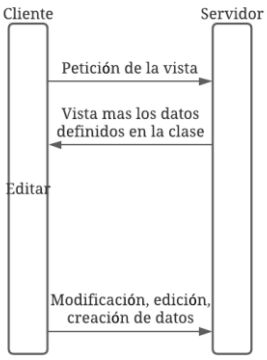


Fig. 18. Proceso de petición de las vistas en Odoo.

Elaborado por el investigador.

Basado en esto, las vistas de Odoo del lado del cliente se muestran en la Fig. 19.

Fig. 19. Vistas en Odoo.

Elaborado por el investigador.

Las vistas en Odoo se generan gracias a los archivos dentro de la carpeta "views" dentro del módulo de Odoo. Cada archivo se explica a continuación.

Archivo "views.xml"

El archivo "*views.xml*" permite mostrar los campos de los usuarios que tiene el modelo por defecto "*res.partner*".

```
<xpath expr="//field[@name='type']" position="before">
    <field name="edad" />
    <field name="paciente"/>
    <field name="dispositivo" />
    <field name="activo" />
</xpath>
```

Donde:

- Edad: es el campo donde se almacena la edad del paciente.

- Paciente: es el campo que muestra si es un paciente o no.
- Dispositivo: en este campo se guardar el id único que posee cada dispositivo Android TV, logrando asociar solo a los usuarios creados desde ese dispositivo.
- Activo: indica si el usuario ha sido borrado desde la aplicación, evitando su renderización en la aplicación, pero conservando sus datos en la nube.

Dentro de este archivo se añaden también dos botones que permitirán redirigir al usuario a la gráfica y mostrar el formulario para enviar prescripciones.

```
<xpath expr="//sheet/div[@name='button_box']" position="inside">
  <button name="button_ver_grafica" type="object" class="oe_stat_button" icon="fa-area-chart">
    <span class="o_field_widget">Ver Gráfica</span>
  </button>

  <button name="button_prescripcion" type="object" class="oe_stat_button" icon="fa-envelope">
    <span class="o_field_widget">Prescripción</span>
  </button>
</xpath>
```

Se añade también una tabla que contenga los datos del paciente según su fecha y hora en las que se hayan subido desde la aplicación. Estos datos se mostrarán de color rojo en el caso de que sean anormales, cuando el nivel de SpO2 sea menor a 95, y el nivel de BPMs cuando son mayores a 100 y menores a 60.

```
<xpath expr="//notebook" position="inside">
  <page string="Historial de signos Vitales">
    <field name="res_paciente_ids">
      <tree editable="bottom" default_order="create_date desc" decoration-danger="(presion&gt;100) or (presion&lt;60) or (oxigeno&lt;95)">
        <field name="create_date" string="Fecha" />
        <field name="presion"/>
        <field name="oxigeno"/>
      </tree>
    </field>
  </page>
</xpath>
```

La vista que se ha definido en esta sección se muestra en la Fig. 20.

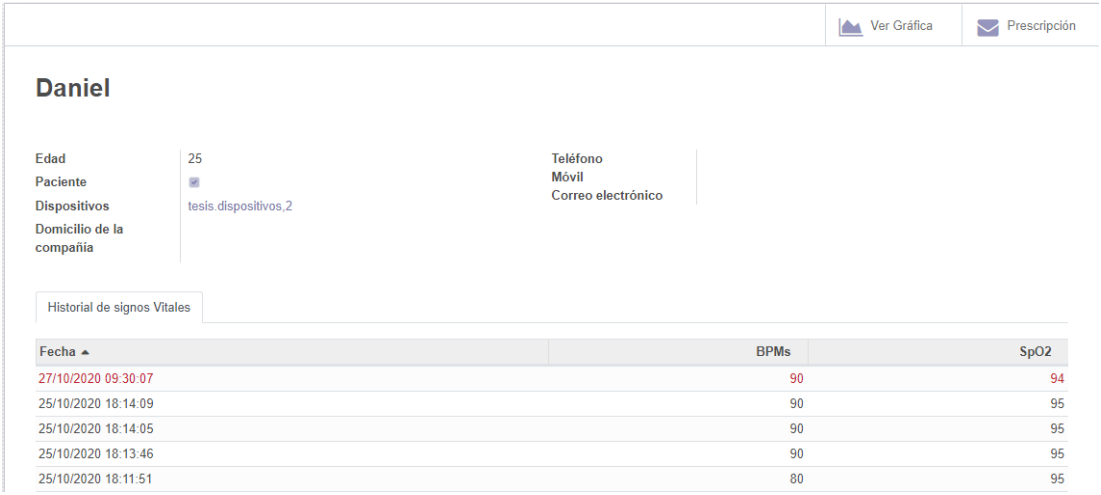


Fig. 20. Vista del usuario en Odoo.

Elaborado por el investigador.

La gráfica se llama a partir del usuario que se ha seleccionado antes, para ello se escribe un filtro mediante el siguiente código.

```
<group string="Group By">
  <filter string="Paciente" name="groupby_res_paciente" domain="[('res_paciente.paciente', '=', True)]" context="{'group_by': 'res_paciente'}"/>
  <filter string="Creado en" name="groupby_create_date" domain="[]" context="{'group_by':'create_date:day'}"/>
  <filter string="Oxigeno" name="groupby_oxigeno" domain="[]" context="{'group_by':'oxigeno'}"/>
</group>
```

Además, se muestran los datos según el día en que estos han sido creados dentro del gráfico. Esta vista se encuentra dentro del archivo "*res_paciente.xml*". El código completo del archivo "*views.xml*" se puede encontrar en el ANEXO D.

```
<graph string="Signos Vitales" orientation="vertical" stacked="False" type="bar" interval="day">
  <field name="res_paciente" group="True"/>
  <field name="presion" type="measure" />
  <field name="oxigeno" type="measure" />
  <field name="create_date" group="True" />
</graph>
```

La vista al presionar el botón "gráfico" se aprecia en la Fig. 21.

Fig. 21. Vista de gráficas según el usuario seleccionado.

Elaborado por el investigador.

Archivo “notificaciones.xml”

Esta vista mostrará una tabla que contiene las notificaciones que se guardan cuando los valores de BPMs y SpO2 son anormales y necesitan ser atendidos. Dentro de este archivo se llaman a los campos dentro del modelo “*tesis.notificaciones*” de la clase “*class Notificaciones*”.

```
<record model="ir.ui.view" id="view_tesis_notificaciones_tree">
    <field name="name">tesis.notificaciones.tree</field>
    <field name="model">tesis.notificaciones</field>
    <field name="arch" type="xml">
        <tree decoration-danger="atendido == False" decoration-success="atendido == True">
            <field name="name" />
            <field name="nombre_paciente"/>
            <field name="presion" />
            <field name="oxigeno" />
            <field name="create_date" string="Fecha" />
            <field name="atendido"/>
        </tree>
    </field>
</record>
```

Donde:

- Name: es el nombre único de cada notificación que se generará automáticamente cada vez que se crea una nueva.
- Nombre_paciente: es el nombre del paciente al que pertenece dichos valores de BPMs y SpO2.
- Presión: es el valor de BPMs que se subieron desde la aplicación y que ha generado la notificación.

- Oxigeno: es el valor de SpO2 que se subieron desde la aplicación y que ha generado la notificación.
- Create_date: es la fecha y hora en la cual esta notificación ha sido guardada en la tabla.
- Atendido: es el campo que permite identificar si dicha notificación ha sido atendida o no.

La vista se muestra en la Fig. 22.

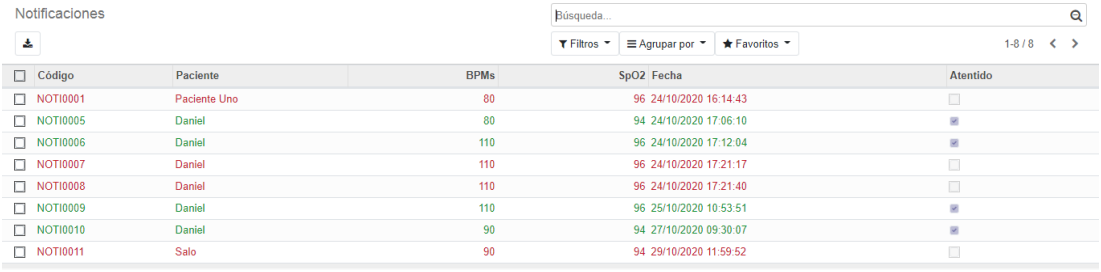


Fig. 22. Vista de notificaciones.

Elaborado por el investigador.

Una vez que se ha seleccionado una notificación para atenderla, se mostrará otra vista con más detalles, mostrando el nombre del paciente, valores de signos vitales y si ha sido atendida o no la notificación. Además, se mostrará un botón que permitirá enviar una prescripción al paciente y cuando esta función sea ejecutada, la notificación pasará al estado de "*atendida*" de forma automática. El código completo del archivo "notificaciones.xml" se muestra en el ANEXO E. La vista finalizada se muestra en la Fig. 23.

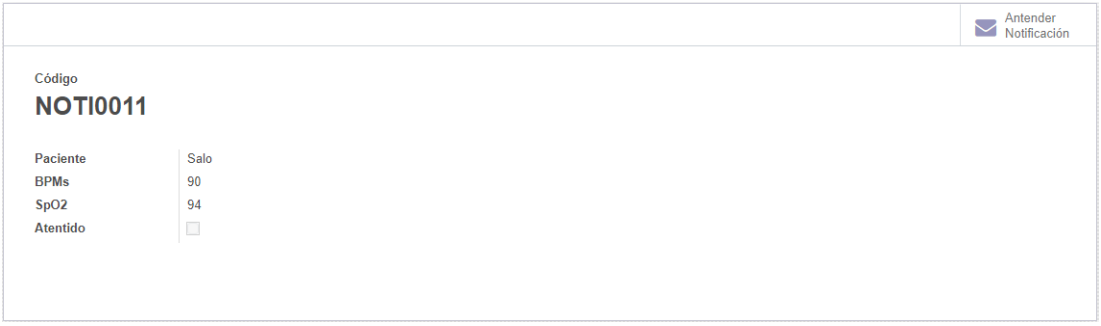


Fig. 23. Vista de detalles de notificación a ser atendida.

Elaborado por el investigador.

Archivo “sequence.xml”

Este archivo da el nombre a las notificaciones y aumentara su número a medida que el usuario del grupo “monitores” las vaya creando. El campo “*prefix*” es el nombre de la notificación y “*padding*” es la cantidad de números que el nombre de la notificación va a tener. El campo “*number_next*” se refiere a la cantidad que se va a añadir al número de la notificación cada vez que se crea una nueva.

```
<record id="id_sequence_tesis_notificaciones" model="ir.sequence">
    <field name="name">Secuencia de Notificaciones</field>
    <field name="code">sequence_tesis_notificaciones</field>
    <field name="prefix">NOTI</field>
    <field name="number_next">1</field>
    <field name="padding">4</field>
    <field name="number_increment">1</field>
    <field name="company_id" eval="False"/>
</record>
```

Archivo “prescripcion.xml”

Este archivo crear una vista que muestra todas las prescripciones enviadas desde la cuenta actual de Odoo dentro de una tabla. El código llama los campos dentro del modelo “*tesis.prescripcion*” de la clase “*class Prescripcion*”. Las columnas que se mostrarán se definen en el código de la siguiente forma.

```
<record model="ir.ui.view" id="view_tesis_prescripcioe_tree">
    <field name="name">tesis.prescripcion.tree</field>
    <field name="model">tesis.prescripcion</field>
    <field name="arch" type="xml">
        <tree default_order="create_date desc" decoration-info="estado == 'draft'" decoration-success="estado == 'send'">
            <field name="res_paciente"/>
            <field name="create_date" string="Fecha" />
            <field name="res_user" groups="base.user_admin" />
            <field name="estado"/>
            <field name="mensaje_leido" />
        </tree>
    </field>
</record>
```

Donde:

- “res_paciente”: es el campo que muestra el nombre del usuario de la aplicación a quien ha sido enviado la prescripción.

- “create_date”: ese el campo que almacena la fecha y hora en la que esta ha sido creada.
- “res_user”: muestra al administrador que usuario de Odoo ha enviado la prescripción.
- “estado”: este campo indica si la prescripción ha sido enviada o solo se ha guardado como un borrador.
- “mensaje_leido”: indica si el usuario de la aplicación a quien ha sido enviada la prescripción ya la ha leído de la aplicación para Android TV.

Esta vista se muestra en la Fig. 24.

Fig. 24. Vista de prescripciones.
Elaborado por el investigador.

Al abrir cada prescripción se podrá modificar en el caso de que sea un borrador, caso contrario esta no puede ser modificada o borrada a menos que sea el administrador quien ejecute la orden. La vista se muestra en la Fig. 25. El código completo del archivo “*prescripcion.xml*” se muestra en el ANEXO F.

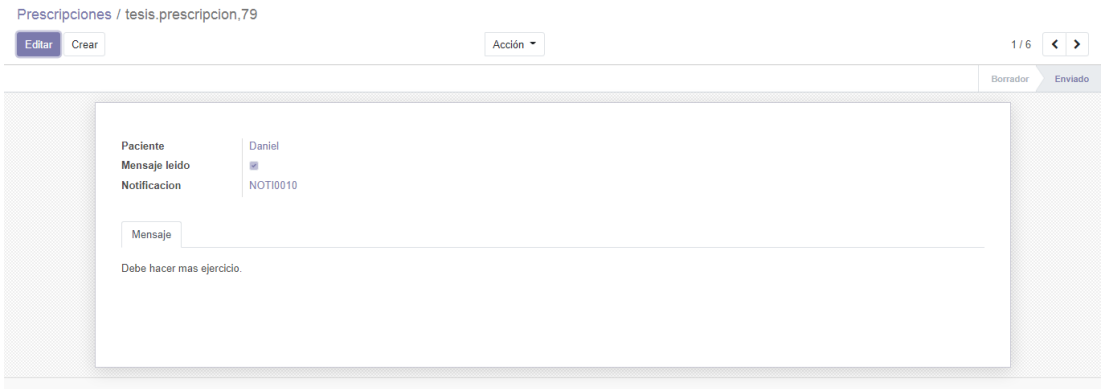


Fig. 25. Vista de la prescripción enviada.
Elaborado por el investigador.

Archivo “dispositivos.xml”

Este archivo llama al campo “*código*” del modelo “*tesis.dispositivos*” para mostrarlo en forma de tabla.

```
<record model="ir.ui.view" id="view_tesis_dispositivos_tree">
    <field name="name">tesis.dispositivos.tree</field>
    <field name="model">tesis.dispositivos</field>
    <field name="arch" type="xml">
        <tree>
            <field name="codigo"/>
        </tree>
    </field>
</record>
```

Esta vista muestra al administrador todos los dispositivos que usan la aplicación para Android TV. La vista se muestra en la Fig. 26. El código completo del archivo “*dispositivos.xml*” se puede ver en el ANEXO G.

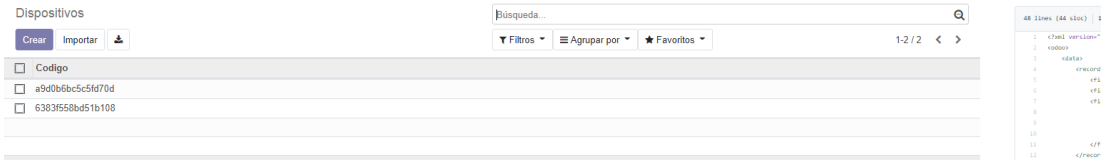


Fig. 26. Vista de los dispositivos en Odoo.

Elaborado por el investigador.

Archivo “controllers.py”

Con los campos para cada paciente que han sido creados, ahora es necesario que la base de datos pueda interactuar con las capas de sensado y de aplicación. Las APIs permitirán esta interacción por medio de peticiones HTTP hacia direcciones específicas dentro del servidor. Estas peticiones se las realiza a partir de textos JSON, alojados en diferentes direcciones de acuerdo con su funcionalidad. Este archivo se encuentra en la dirección “*/odoo/custom/custom-addons/tesis/controllers/controllers.py*”. Los comandos utilizados para la implementación de las APIs se explican en la tabla 15.

TABLA XV. Comandos utilizados para la implementación de las APIs.

Elaborado por el investigador.

Comando	Función
@http.route	Esta función crea rutas web dentro del servidor para comunicaciones de tipo HTTP. Se puede definir el tipo de método (GET, POST, etc.) y el texto que se va a mostrar (HTML, JSON, etc.)
request.env	Es un objeto estándar que contiene información importante de los datos almacenados en Odoo. Puede ser utilizada para consultar la información de todos los usuarios mediante el objeto "res.partner".
sudo()	Es una función que permite ejecutar código como super usuario, dando acceso a toda la información almacenada en la base de datos de Odoo.
search()	Busca información específica de un objeto dentro de Odoo. Esta solo puede ser ejecutada como super usuario, con el comando "sudo()".
append()	Es una función que añade contenido al final de un elemento u objeto.
request.httprequest.data	Es un objeto que contiene información de la petición HTTP realizada a una dirección web cuando se utiliza texto que no sea de tipo HTML.
json.loads()	Es una función que recoge el contenido de un texto en una petición HTTP y lo transforma en un texto de tipo "string".
self._response()	Genera una respuesta web a una petición realizada con un tipo de archivo JSON.

decode()	Es un método de tipo string el cual decodifica el texto asignado en el codec definido.
write()	Es una función que sobrescribe un texto de tipo string en un texto asignado.
create()	Es una función que crea valores de tipo string en un objeto seleccionado.
request.env.uid	Es un objeto que contiene la información del "id" del usuario definido.
ref()	Es una función que toma un string utilizando un "id" externo y devuelve un registro o un archivo.
unlink()	Es un método que maneja el borrado de datos en función de un "id" asignado.

El código completo se puede ver en el ANEXO H. Debido a su tamaño solo se explicarán las funcionalidades de cada API. El archivo contiene 6 APIs, una para cada funcionalidad del sistema de telemedicina.

"/api/datos/idDispositivo"

Esta API busca todos los usuarios activos de cada dispositivo que han sido creados para mostrarlos en un archivo JSON en la dirección web "*http://18.222.17.116:8069/api/datos/idDispositivo*". Esta permitirá a la capa de aplicación obtener los datos de los usuarios, para mostrar sus nombres en la interfaz de la aplicación al pasarle el campo "*idDispositivo*" único de cada televisión Android TV. Esta API se encuentra en la función llamada "*get_data*" en el archivo "*controllers.py*". Las peticiones HTTP se realizan mediante el método GET.

"/api/paciente/id"

Al enviar el parámetro "*id*" del paciente a consultar, esta API devuelve el nombre, edad e id del mismo. Este servirá para consultar cambios que se hayan hecho al paciente durante la navegación entre las diferentes ventas que posee la aplicación. Esta

se encuentra en la misma función que la API anterior, solo que se condicionan las peticiones en función de los campos "*id*" del dispositivo y del usuario.

"/api/crear_usuario"

Aquí se obtienen los datos enviados que el usuario a ingresado desde la aplicación por medio de un método POST y un texto de tipo JSON. Esta API se encuentra en la dirección web "*http://18.222.17.116:8069/api/crear_usuario*". Si el usuario ya ha sido creado, esta API evitara la creación de un usuario duplicado modificando el campo "*activo*" del modelo "*ResParter*" a "*True*".

"api/modificar_usuario"

Una vez creado un usuario, los campos pueden ser modificados mediante esta API. Los datos de los usuarios pueden ser modificados desde la interfaz gráfica de la aplicación. La aplicación envía los datos modificados en texto JSON mediante el método PUT a la dirección web "*http://18.222.17.116:8069/api/modificar_usuario*".

Estas dos API están dentro de la función "*create*" del archivo "*controllers.py*". Se realiza un condicional para saber el tipo de método enviado. Si es PUT se ejecuta "*/api/modificar_usuario*", caso contrario "*/api/crear_usuario*". Por último, se recibe el id tanto del usuario como del dispositivo para poder realizar la búsqueda y modificación del usuario.

"api/borrar"

Desde la interfaz gráfica de la aplicación el usuario envía el "*id*" del usuario seleccionado mediante un texto JSON con el método POST a la dirección web "*http://18.222.17.116:8069/api/borrar"*. La API busca el usuario mediante la "*id*" recibida y lo modifica el campo "*activo*" del modelo "*ResPartner*" a "*False*". Este proceso evita que el usuario sea mostrado en la API "*/api/datos/idDispositivo*". La API está dentro de la función "*borrar*" del archivo "*controllers.py*".

"/api/splash"

Esta API se ejecuta cada vez que se inicia la aplicación con el fin de verificar si el id del dispositivo Android TV esta registrado dentro de la base de datos que maneja el

modelo "*tesis.dispositivos*". Si el id del dispositivo ya existe, esta API no ejecuta ninguna acción, caso contrario crea un nuevo dispositivo en la base de datos.

"/api/datosPaciente"

Esta recibe los datos de BPMs y SpO2 de la aplicación una vez que hayan sido medidos. Al recibirlos los compara en caso de que sean anormales y muestra una advertencia en la página web de Odoo como se muestra en la Fig. 27.

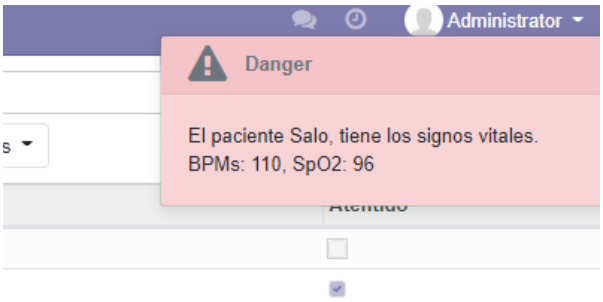


Fig. 27. Notificación de advertencia cuando el paciente presenta signos anormales. Elaborado por el investigador.

Las condiciones de activación de la notificación es cuando el nivel de BPM es menor a 60 o mayor a 100, y el de SpO2 cuando es menor a 95%.

"/api/prescripción/idPaciente"

Esta carga la última prescripción enviada al usuario desde la página de Odoo para poder mostrarlas en la aplicación. Busca la prescripción en la base de datos del modelo "*tesis.prescripcion*" en función del id que se le ha enviado. Si no encuentra ninguna prescripción enviara el mensaje "No se ha encontrado una prescripción".

"/api/grafica"

Esta API va a construir el objeto que la gráfica necesita para mostrar los datos, filtrándolos en función de las fechas que se haya recibido desde la aplicación. Además, esta mostrará todas las prescripciones que se hayan enviado desde Odoo filtrándolas de igual manera que los datos. Por último, esta cuenta todas las prescripciones que no hay sido leídas para mostrarlas en la aplicación. Se construye un objeto con todos esto valores que serán enviados a la aplicación, y para que puedan ser renderizados.

"/api/actNoti"

Esta API recibe el "*id*" del usuario y modifica el campo "*mensaje_leido*" del modelo "*tesis.prescripcion*" cuando el usuario haya visto la prescripción que ha sido enviada desde Odoo.

Security

Esta carpeta contiene dos archivos diferentes:

- **Control de acceso:** en este archivo se pueden especificar los accesos que tiene cada grupo de usuario a los modelos y sus vistas en un archivo "*ir.model.access.csv*". Para este proyecto se ha creado un grupo llamado "monitor", el cual se encarga de enviar las prescripciones y consultar los signos de los usuarios. Sus permisos dependerán de las funciones de cada modelo. Estos permisos se especifican en la Fig. 28.

id	name	model_id:id	group_id:id	perm_read	perm_write	perm_create	perm_unlink
access_tesis_res_paciente	access_tesis_res_paciente	model_res_paciente	base.user_admin	1	1	1	1
access_tesis_prescripcion	access_tesis_prescripcion	model_tesis_prescripcion	base.user_admin	1	1	1	1
access_tesis_dispositivos	access_tesis_dispositivos	model_tesis_dispositivos	base.user_admin	1	1	1	1
access_tesis_notificaciones	access_tesis_notificaciones	model_tesis_notificaciones	base.user_admin	1	1	1	1
access_tesis_res_partner_monitor	access_tesis_res_partner_monitor	model_res_partner	tesis.group_monitor_tesis	1	0	0	0
access_tesis_res_paciente_monitor	access_tesis_res_paciente_monitor	model_res_paciente	tesis.group_monitor_tesis	1	0	0	0
access_tesis_prescripcion_monitor	access_tesis_prescripcion_monitor	model_tesis_prescripcion	tesis.group_monitor_tesis	1	1	1	0
access_tesis_notificaciones_monitor	access_tesis_notificaciones_monitor	model_tesis_notificaciones	tesis.group_monitor_tesis	1	0	0	0
access_tesis_dispositivos_monitor	access_tesis_dispositivos_monitor	model_tesis_dispositivos	tesis.group_monitor_tesis	1	0	0	0

Fig. 28. Permisos para acceder a los modelos en Odoo.

Elaborado por el investigador.

Este grupo solo puede leer datos de todos los modelos, a excepción del modelo de "*tesis.prescripcion*", el cual puede crear y modificar los mismos ya que estas funciones son necesarias para el envío de prescripciones desde una cuenta que no sea del administrador, el cual tiene acceso a todas las funciones de seguridad.

- **Reglas de registro:** estas son básicamente reglas para ejecutar operaciones como crear o actualizar datos de los distintos modelos. Estas reglas están dentro del archivo "*security.xml*". En este archivo se crea el grupo "Monitores" que estarán a cargo de enviar prescripciones una vez analizados los datos subidos.

```
<record id="group_monitor_tesis" model="res.groups">
    <field name="name">Monitores</field>
</record>
```

También se bloquean las vistas de administrador a usuarios de Odoo que no estén dentro del grupo llamado "*base.user_admin*" que son las cuentas de Odoo que son administradoras.

```
<record model="ir.ui.menu" id="website.menu_website_configuration">
    <field name="groups_id" eval="[(6, 0, [ref('base.user_admin')])]"/>
</record>

<record model="ir.ui.menu" id="mail.menu_root_discuss">
    <field name="groups_id" eval="[(6, 0, [ref('base.user_admin')])]"/>
</record>
```

Por último, se filtran las prescripciones enviadas solo por la cuenta activa en Odoo.

```
<data noupdate="1">
    <record model="ir.rule" id="tesis_prescripcion_rule">
        <field name="name">tesis.prescripcion: leer prescripcion por usuario logeado</field>
        <field name="model_id" ref="tesis.model_tesis_prescripcion"/>
        <field name="domain_force">[('res_user', '=', user.id)]</field>
        <field name="groups" eval="[(4, ref('group_monitor_tesis'))]"/>
    </record>
</data>
```

Una vez implementadas todas las vistas de Odoo se debe instalar la aplicación desde la sección de "Aplicaciones" en Odoo.

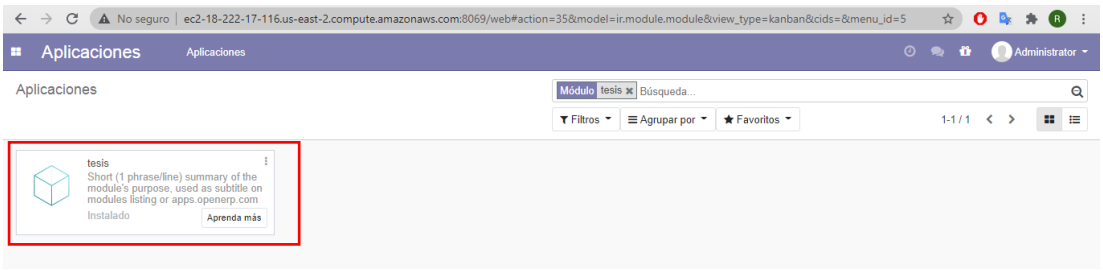


Fig. 29. Instalación del módulo de Odoo.

Elaborado por el investigador.

Para que todos archivos de vistas se puedan mostrar correctamente se debe detener el servidor de Odoo con el comando "*sudo service odoo-server stop*".

```
ubuntu@ip-172-31-23-168:~$ sudo service odoo-server stop
```

Después entrar al servidor de Odoo con el comando "*sudo su odoo*" y actualizar la aplicación con el comando "*/odoo-server/odoo-bin -u tesis -c /etc/odoo-server.conf*".

```
ubuntu@ip-172-31-23-168:~$ sudo su odoo
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

odoo@ip-172-31-23-168:/home/ubuntu$ ~/odoo-server/odoo-bin -u tesis -c /etc/odoo-server.conf
```

Entonces se ejecuta el comando "*exit*" y se reinicia el servidor de Odoo mediante el comando "*sudo service odoo-server start*".

Para finalizar, desde la cuenta de administrador se instala la aplicación "*Sitio web*" desde la página de "*Aplicaciones*" como se muestra en la Fig. 30. para poder modificar la página de inicio de Odoo.

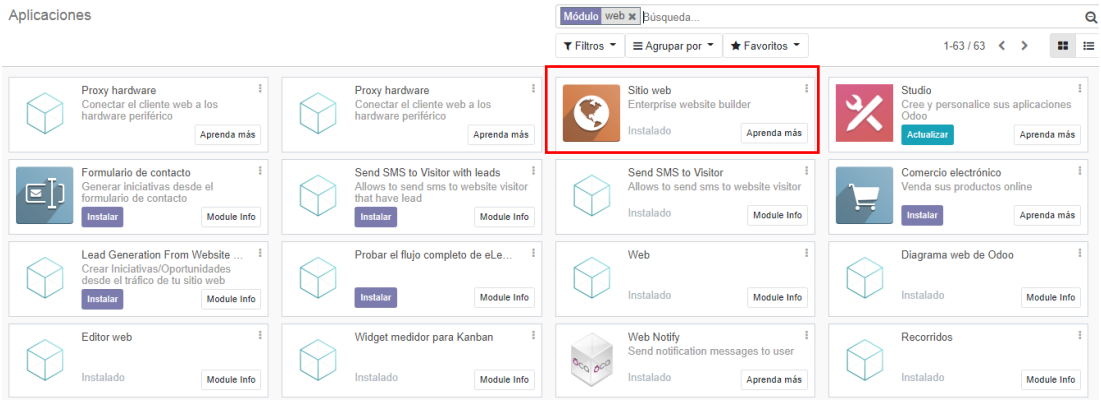


Fig. 30. Instalación de la aplicación "Sitio web" en Odoo.

Elaborado por el investigador.

Al modificar la página principal de Odoo, esta se mostrará como en la Fig. 31.

Fig. 31. Página principal del Odoo.

Elaborado por el investigador.

Implementación del WebSocket

Este permitirá una comunicación "full-duplex" entre un cliente y servidor mediante un protocolo TCP, con el fin de informar al usuario que ha recibido una prescripción. Se ha optado por esta opción ya que en el sistema operativo Android TV no existe una barra de estado como lo hay en los teléfonos móviles, una funcionalidad que les permite mostrar notificaciones. En Android TV, se mostrarán mensajes de tipo "Toast" cuando una prescripción ha sido enviada desde Odoo usando una conexión WebSocket, permitiendo mostrarlos aun si la aplicación está ejecutándose en segundo plano.

El código del servidor WebSocket está escrito en Python y está alojado en la dirección "*/opt*" del directorio de Linux de la maquina Ubuntu del servicio AWS. Primero se añaden a todos los clientes que se conectados mediante el protocolo WebSocket a una variable llamada "*clients*", y se los borra en caso de desconexión.

```
async def register(self, ws: WebSocketServerProtocol) -> None:
    self.clients.add(ws)
    logging.info(f'{ws.remote_address} connects.')

async def unregister(self, ws: WebSocketServerProtocol) -> None:
    self.clients.remove(ws)
    logging.info(f'{ws.remote_address} disconnects.')
```

Luego se crea una función "*send_clients*" en la cual se envía el mensaje que es pasado del modelo "*tesis.prescripcion*" mediante el objeto "*data*". Este luego se llama desde la función "*distribute*" que es el encargado de enviar los mensajes mediante el protocolo WebSocket, pasando la variable "*message*" que va a ser enviada a todos los usuarios conectados.

```
async def send_clients(self, message: str) -> None:
    if self.clients:
        await asyncio.wait([client.send(message) for client in self.clients])

async def distribute(self, ws: WebSocketServerProtocol) -> None:
    async for message in ws:
        await self.send_clients(message)
```

Estas funciones se manejan desde la función "*ws_handler*" la cual envía los mensajes de forma continua, o desconecta a los usuarios en caso de que se haya perdido la conexión.

```
async def ws_handler(self, ws: WebSocketServerProtocol, uri: str) -> None:
    await self.register(ws)
    try:
        await self.distribute(ws)
    finally:
        await self.unregister(ws)
```

Para finalizar se define el puerto de comunicación estándar del protocolo WebSocket que es el 80, se define el servidor en la dirección IP interna con "*0.0.0.0*" y se manda a ejecutar en un bucle infinito.

```
server = Server()
start_server = websockets.serve(server.ws_handler, "0.0.0.0", 80)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

Al finalizar, se ejecuta como un script en segundo plano con el comando "*nohup python3 tesis-socket.py> tesis-socket.out &*" donde "*tesis-socket.py*" es el nombre del archivo que se asignó al código antes escrito. El archivo "*tesis-socket.out*" es un log de ejecución del archivo "*tesis-socket.py*", útil para la identificación de errores en su ejecución.

La aplicación realizada en esta sección se explica mediante el gráfico UML de caso de uso mostrado en la Fig. 32.

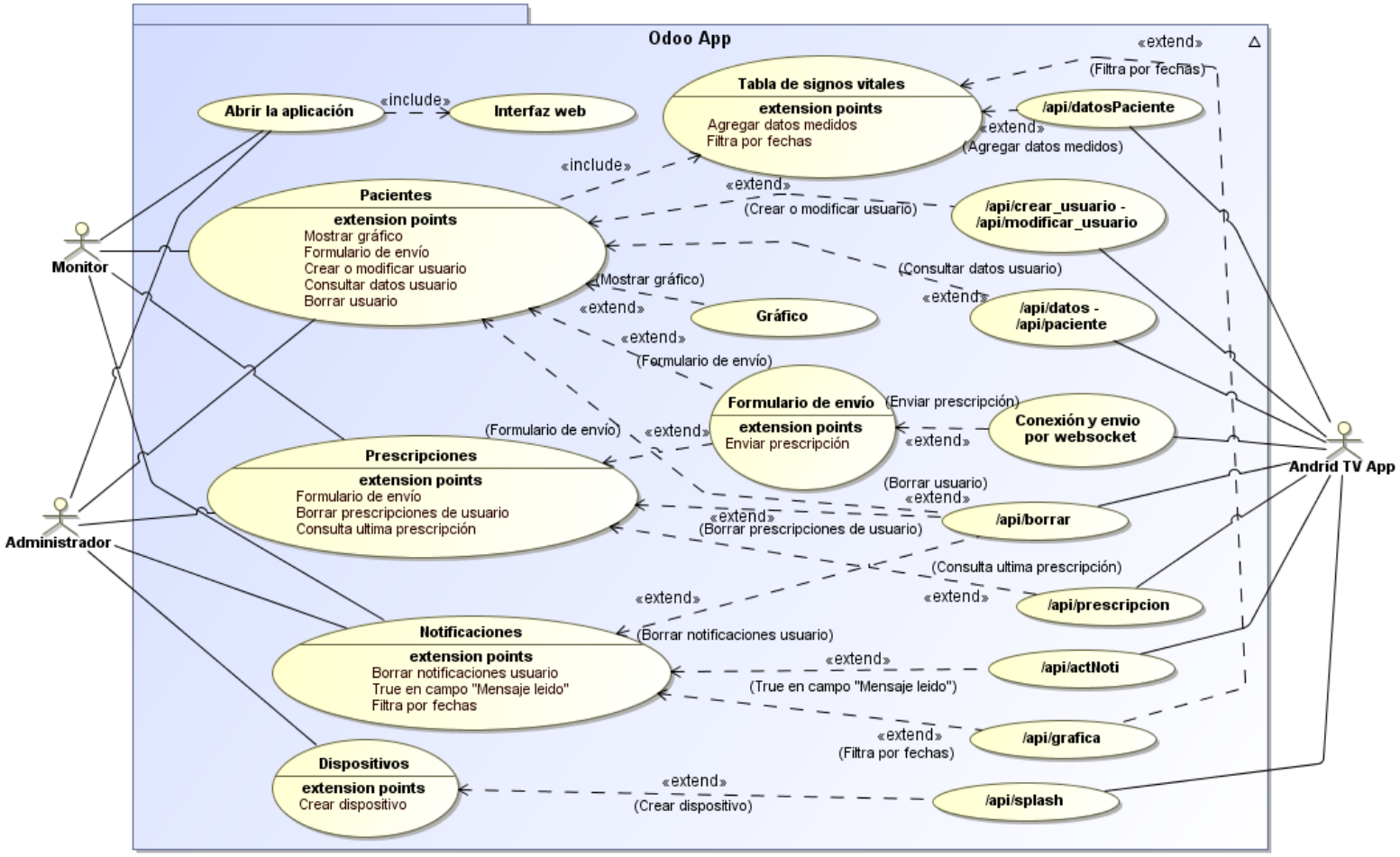


Fig. 32. Diagrama UML de caso de uso del funcionamiento de la aplicación para Odoo.

Elaborado por el investigador.

3.1.1.4. Implementación del circuito de monitoreo de signos vitales IoT.

3.1.1.4.1. Acondicionamiento del sensor MAX30100

El sensor MAX30100 es un sensor que monitorea el pulso cardiaco. Este trabaja con un fotodiodo que emite luz infrarroja (950 nm) y un diodo que emite luz roja (650 nm). El sensor puede ser ubicado en el lóbulo de la oreja, en el dedo o en algún lugar donde la piel es suficientemente delgada para que estas longitudes de onda puedan penetrar en la piel, como en la muñeca en la parte baja del pulgar como en la Fig. 33. Esto permitirá el diseño de una pulsera Weareable.

Fig. 33. Pulso en la muñeca.
Elaborado por el investigador.

Cuando el corazón late, la absorción de la luz roja por el fotodiodo varia debido al flujo sanguíneo. Esta variación puede ser utilizada para calcular el nivel de oxígeno que existe en la Hemoglobina (Hb), las cuales son moléculas que contienen oxígeno y son las que la transportan por nuestra sangre. El principio de funcionamiento de este sensor se explica en la Fig. 34.

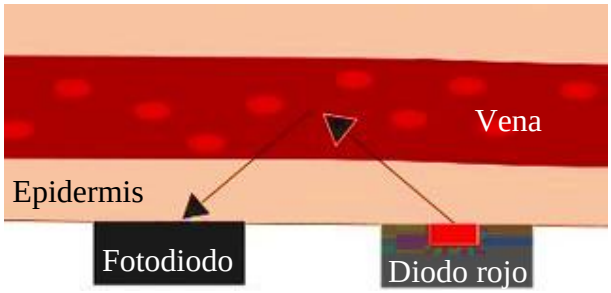


Fig. 34. Principio de funcionamiento del sensor MAX30100.
Elaborado por el investigador.

Para el funcionamiento de este sensor se utilizará la librería para Arduino "*MAX30100.h*". La cual fue implementada por el usuario de GitHub "*OXullo Intersecans".* Esta librería lee los datos en crudo del fotodiodo del sensor, los cuales son como la Fig. 35.

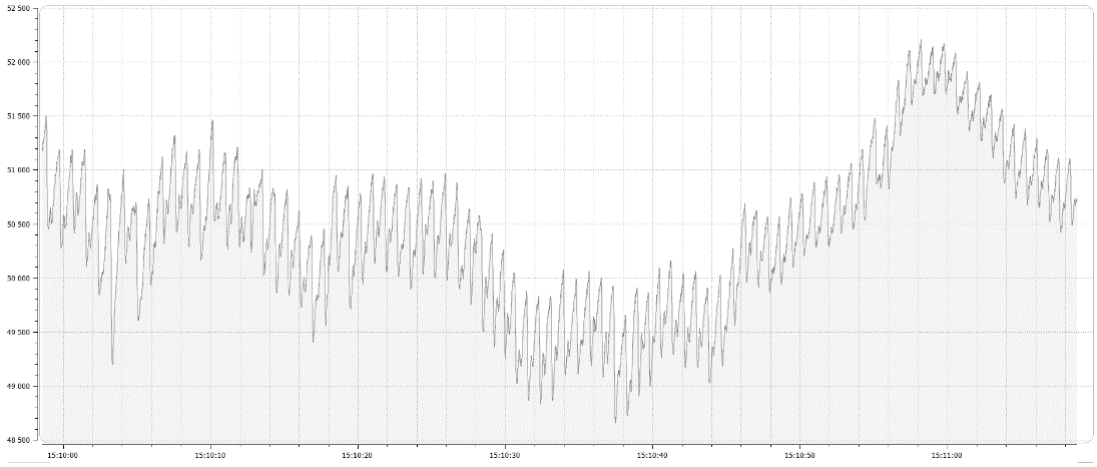


Fig. 35. Datos del fotodiodo [41].

Esta gráfica muestra que los datos oscilan ligeramente, y que existe una componente DC. Para poder leer correctamente el pulso cardiaco y el nivel de oxígeno en la sangre es necesario eliminar esta componente. Para esto se implementa un filtro IIR, el cual atenuará la banda estrecha de la componente DC, usando (1) y (2).

$$w(t)=x(t)+\propto w(t-1) \quad \textbf{(1)}$$

$$y(t)=w(t)-w(t-1) \quad \textbf{(2)}$$

Donde *y(t)* es la salida del filtro, *x(t)* es la muestra actual que entra al filtro, *w(t)* es donde se guardan los valores anteriores de la componente DC y $\propto$ es el encargado de ampliar o reducir el filtro. Si $\propto$ es 1 el filtro dejara pasar todo, si es 0 el filtro no dejara pasar nada. Lo ideal es definir un valor cercano a 1, en esta librería se define el valor 0.95, como a continuación.

```
#define DC_REMOVER_ALPHA                0.95
```

Una vez aplicado este filtro, la salida de este será como lo indica la Fig. 36.

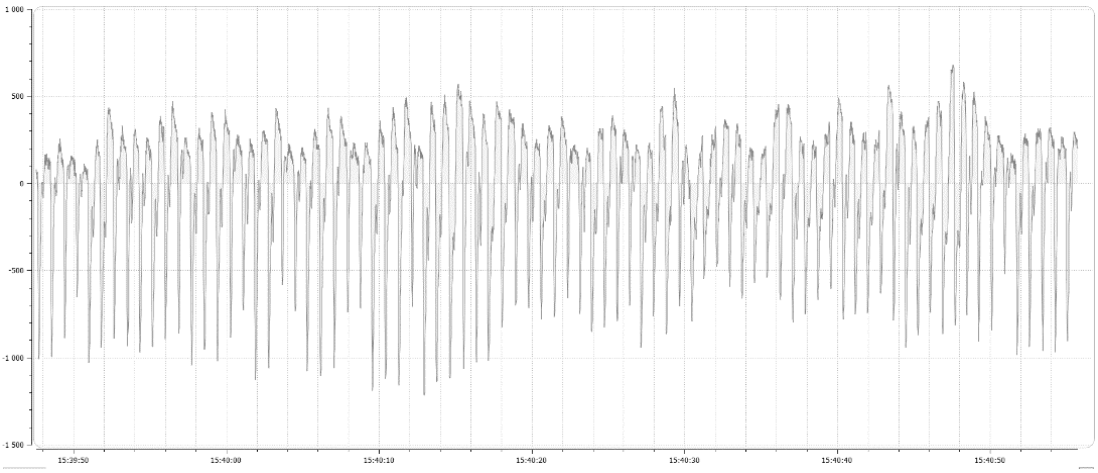


Fig. 36. Salida del filtro IIR [41].

En la Fig. 51 se muestra que el componente DC esta alrededor del valor 50000, en la Fig. 52 los valores oscilan entre el valor 0, donde se aprecia que la componente DC ha sido eliminada. El código fuente del filtro implementado, es como se muestra a continuación.

```
class DCRemover
{
public:
    DCRemover() : alpha(0), dcw(0)
    {
    }
    DCRemover(float alpha_) : alpha(alpha_), dcw(0)
    {
    }

    float step(float x)
    {
        float olddcw = dcw;
        dcw = (float)x + alpha * dcw;

        return dcw - olddcw;
    }

    float getDCW()
    {
        return dcw;
    }

private:
    float alpha;
    float dcw;
};
```

La señal de salida del filtro posee armónicos de frecuencias altas, para eliminarlas se debe implementar un filtro Butterworth pasa bajo. Para ello se toma en cuenta dos variables: Fs y Fc. En algunos casos los batidos por minuto de un paciente pueden llegar a 220 BPMs (batidos por minuto), lo cual nos daría una frecuencia de:

$$f=\frac{220\ \text{bpm}}{60}=3.66\ \text{Hz} \quad \textbf{(3)}$$

Y suponiendo que la medición más baja sea 50 BPMs:

$$f=\frac{50\ \text{bpm}}{60}=0.83\ \text{Hz} \quad \textbf{(4)}$$

De acuerdo con la tabla 17, sacada del documento del fabricante del sensor, el ancho de pulso y la frecuencia de muestreo están ligadas, dando como resultado la resolución. La librería opta por seleccionar una resolución de 16 bits, 100 Hz de frecuencia de muestreo y 1600 us (microsegundos) de ancho de pulso. De esta forma se aplica la mayor resolución y frecuencia posibles.

TABLA XVI. Relación entre el ancho de pulso y la frecuencia de muestreo [42, p. 19].

Frecuencia de muestreo (Hz)	**Ancho de pulso (us)**			
	200	**400**	**800**	**1600**
50	o	o	o	o
100	o	o	o	o
167	o	o	o	-
200	o	o	o	-
400	o	o	-	-
600	o	o	-	-
800	o	o	-	-
1000	o	o	-	-
Resolución (bits)	**13**	**14**	**15**	**16**

Debido a que la mayor frecuencia presente en la señal podría ser 3.66 Hz, la frecuencia de corte del filtro se podría definir en 4 Hz. Pero el filtro podría cortar señales útiles para la medición, por lo que en la librería se define una frecuencia de 10 Hz para Fc, siendo esta suficiente para limpiar y mejorar la señal lo necesario como para medir los picos. La señal de salida del filtro se muestra en la Fig. 37.

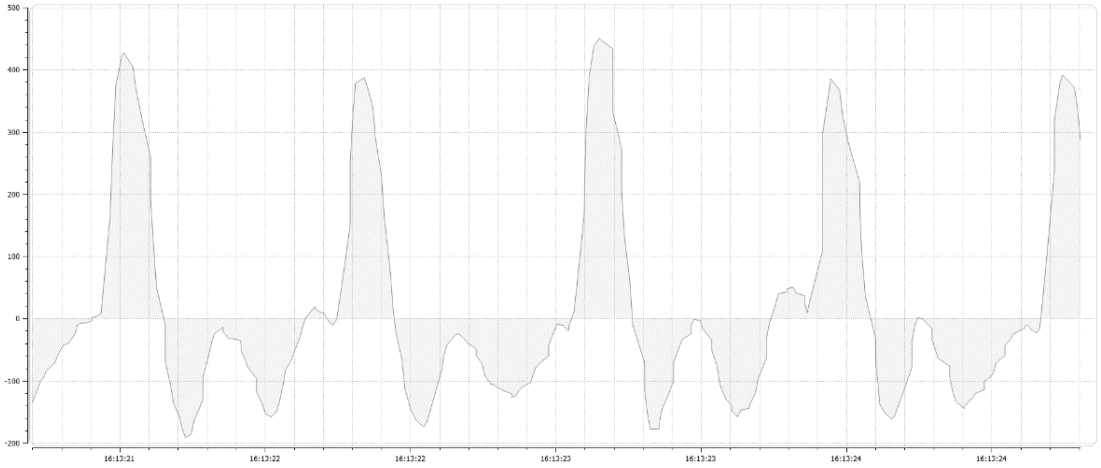


Fig. 37. Salida del filtro Butterworth [41].

De esta forma el filtro dejara pasar las frecuencias deseadas. La implementación del filtro se muestra a continuación.

```
class  FilterBuLp1
{
    public:
        FilterBuLp1()
        {
            v[0]=0.0;
        }
    private:
        float v[2];
    public:
        float step(float x) //class II
        {
            v[0] = v[1];
            v[1] = (2.452372752527856026e-1 * x)
                 + (0.50952544949442879485 * v[0]);
            return
                 (v[0] + v[1]);
        }
};
```

Detección de BPMs

Para la detección de las pulsaciones por minuto, se establece una máquina de estados, la cual leerá continuamente las muestras tomadas del sensor. La máquina permitirá calcular las BPMs por medio de la ecuación (5).

$$BPM=\frac{60000}{\text{perido de pulso}} \quad \textbf{(5)}$$

La máquina de estados implementada se muestra en la Fig. 38.

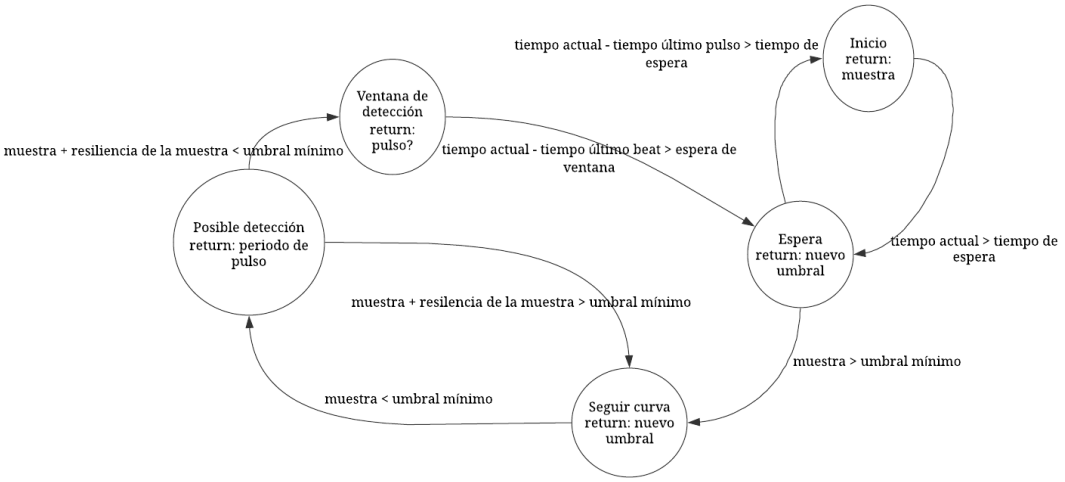


Fig. 38. Máquina de estados para la detección y cálculo de BPMs.

Elaborado por el investigador.

Variables del programa

Antes de comenzar con el análisis es necesario definir las variables que se utilizarán para obtener los BPMs, estas se definen en el archivo "*MAX30100_BeatDetector.h*". La tabla 18 muestra las variables que este código utiliza. El código completo se puede revisar en el ANEXO I.

TABLA XVII. Variables del programa para el cálculo de BPMs.

Elaborado por el investigador.

Nombre	**Función**	**Valor**
BEATDETECTOR_INIT_HOLDOFF	Amacena el tiempo a esperar antes de realizar una medición.	2000 ms
BEATDETECTOR_MASKING_HOLDOFF	Almacena el tiempo de ventana entre un pulso medido correctamente y otro. Si este se cumple, se detiene la conversión de BPMs y se reinicia la medición.	200 ms
BEATDETECTOR_BPFILTER_ALPHA	Almacena un factor para dar un peso igual de manera exponencial a todos los datos convertidos en BPMs. Ayuda en el filtrado de la curva de pulsos detectados.	0.6
BEATDETECTOR_MIN_THRESHOLD	Valor mínimo permitido que se ha detectado en la curva de pulsos. Este ayuda en la detección de pulsos de baja intensidad.	20
BEATDETECTOR_MAX_THRESHOLD	Valor máximo que se ha detectado en la curva de pulsos. Este ayuda a saber si el pulso está dentro de la curva o es un dato basura. Limita los BPMs a un máximo de 220.	800
BEATDETECTOR_STEP_RESILIENCY	Es el valor máximo negativo al que puede llegar la curva de pulsos. Ayuda a identificar el final de un pulso correcto medido.	30
BEATDETECTOR_INVALID_READOUT_DELAY	Es el valor máximo permitido de tiempo que permanece la maquina esperando a una medición valida de un pulso. Si este se cumple los valores y la maquina se reinician.	2000

Inicio y Espera

Estos estados definen un tiempo de espera antes de empezar a tomar muestras y compararlas con los umbrales de medición. Si el tiempo actual de programa supera el tiempo almacenado en la variable "BEATDETECTOR_INIT_HOLDOFF", el estado "Inicio" pasa al estado de espera, donde se compara la muestra con el umbral de valor más pequeño almacenado en la variable "BEATDETECTOR_MIN_THRESHOLD" y si lo es, saca el mínimo valor del medido y el valor de umbral máximo almacenado en la variable "BEATDETECTOR_MAX_THRESHOLD", y lo cambia con el umbral mínimo para la siguiente medición. Entonces se cambia al estado "Seguir curva". En caso de que no se haya detectado un pulso dentro del valor almacenado en la variable "BEATDETECTOR_INVALID_READOUT_DELAY", los valores y la máquina de estados se reinicia. El código fuente de estos dos estados se muestra a continuación.

```
case BEATDETECTOR_STATE_INIT:
    if (millis() > BEATDETECTOR_INIT_HOLDOFF) {
        state = BEATDETECTOR_STATE_WAITING;
    }
    break;

case BEATDETECTOR_STATE_WAITING:
    if (sample > threshold) {
        threshold = min(sample, BEATDETECTOR_MAX_THRESHOLD);
        state = BEATDETECTOR_STATE_FOLLOWING_SLOPE;
    }

    // Tracking lost, resetting
    if (millis() - tsLastBeat > BEATDETECTOR_INVALID_READOUT_DELAY) {
        beatPeriod = 0;
        lastMaxValue = 0;
    }

    decreaseThreshold();
    break;
```

Seguir curva

Si la nueva muestra es mayor a la anterior, en este caso se tiene un posible pulso detectado, en caso contrario el pulso se vuelve a comparar con el valor máximo almacenado en la variable "BEATDETECTOR_MAX_THRESHOLD", para obtener el mínimo de los dos, donde el nuevo valor de umbral mínimo cambia, y se continua con el siguiente estado. El código del estado se muestra a continuación.

```
case BEATDETECTOR_STATE_FOLLOWING_SLOPE:
    if (sample < threshold) {
        state = BEATDETECTOR_STATE_MAYBE_DETECTED;
    } else {
        threshold = min(sample, BEATDETECTOR_MAX_THRESHOLD);
    }
    break;
```

Posible detección

En este estado se compara si la suma del valor de la muestra y el valor máximo negativo o de resiliencia pueden sobrepasar el valor del umbral mínimo antes cambiado, significando un pulso detectado. Se mide el tiempo actual de ejecución del programa y se lo guarda en la variable "*tsLastBeat*". Cada vez que la diferencia de este tiempo y el tiempo de ejecución del programa es diferente de cero, se calcula el periodo de pulso con la ecuación (6).

$$\text{periodo de pulso}=\text{BEATDETECTOR\_BPFILTER\_ALPHA}*\text{delta} + (1-\text{BEATDETECTOR\_BPFILTER\_ALPHA})*\text{periodo de pulso} \quad \textbf{(6)}$$

Donde delta es igual a la diferencia entre el valor actual de programa y el tiempo en el cual se detectó un pulso. Si la muestra más su valor máximo de pico negativo o de resiliencia no superan al nuevo umbral mínimo, se deberá volver a seguir la curva para saber el nuevo valor y seguir con el cálculo. El código fuente para este estado se muestra a continuación.

```
case BEATDETECTOR_STATE_MAYBE_DETECTED:
    if (sample + BEATDETECTOR_STEP_RESILIENCY < threshold) {
        // Found a beat
        beatDetected = true;
        lastMaxValue = sample;
        state = BEATDETECTOR_STATE_MASKING;
        float delta = millis() - tsLastBeat;
        if (delta) {
            beatPeriod = BEATDETECTOR_BPFILTER_ALPHA * delta +
                    (1 - BEATDETECTOR_BPFILTER_ALPHA) * beatPeriod;
        }

        tsLastBeat = millis();
    } else {
        state = BEATDETECTOR_STATE_FOLLOWING_SLOPE;
    }
    break;
```

Ventana de detección

Cada vez que se detecta un pulso, el estado se cambia a la ventana de detección. Si el tiempo que ha transcurrido entre el ultimo latido y el tiempo actual de ejecución del programa es mayor al valor de la variable "BEATDETECTOR_MASKING_HOLDOFF", significa que no ha ocurrido un pulso y se regresa al estado de "Espera". Este estado sirve para saber si los pulsos son continuos y que se puede continuar con el cálculo del periodo de pulso.

```
case BEATDETECTOR_STATE_MASKING:
    if (millis() - tsLastBeat > BEATDETECTOR_MASKING_HOLDOFF) {
        state = BEATDETECTOR_STATE_WAITING;
    }
    decreaseThreshold();
    break;
```

Todos los códigos fuentes están dentro del archivo "*MAX30100_BeatDetector.cpp*" de la librería. El código completo puede ser encontrado en el ANEXO J.

Cálculo del nivel de oxígeno (SpO2)

Para calcular el nivel de oxígeno en la sangre se mide la amplitud RMS de la señal del led infrarrojo y del diodo rojo. En este caso debe ser la componente AC, ya que es aquella la que refleja la absorción en las arterias. Los valores RMS se calculan simplemente elevando al cuadrado los valores medidos del sensor. La razón R de la ecuación (7) permitirá compararla con la tabla de consulta de la relación de proporción de la Fig. 39, para el cálculo de la oxigenación en la sangre.

$$R=100*\frac{\log(\text{Amplitud RMS de la intensidad led rojo})}{\log(\text{Amplitud RMS de la intensidad del led infrarrojo})} \qquad \textbf{(7)}$$

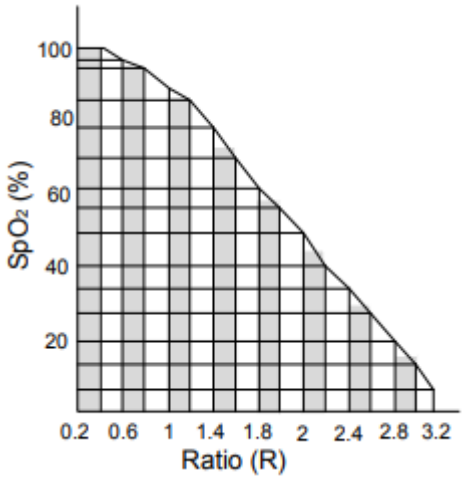


Fig. 39. Tabla de consulta de la relación de proporción para el cálculo de la oxigenación de la sangre [43, p. 3].

Una vez medidos los valores, se aplica la formula y después se comparan con la tabla, obteniendo el nivel de oxígeno en la sangre. El código fuente para el cálculo se encuentra en el archivo "*MAX30100_SpO2Calculator.cpp*" en el ANEXO K de este documento. El cálculo se realiza en el método "*SpO2Calculator*".

En la Fig. 40 se muestra el diagrama de flujo del funcionamiento de la librería MAX30100 para Arduino que se ha utilizado para la implementación de este proyecto.

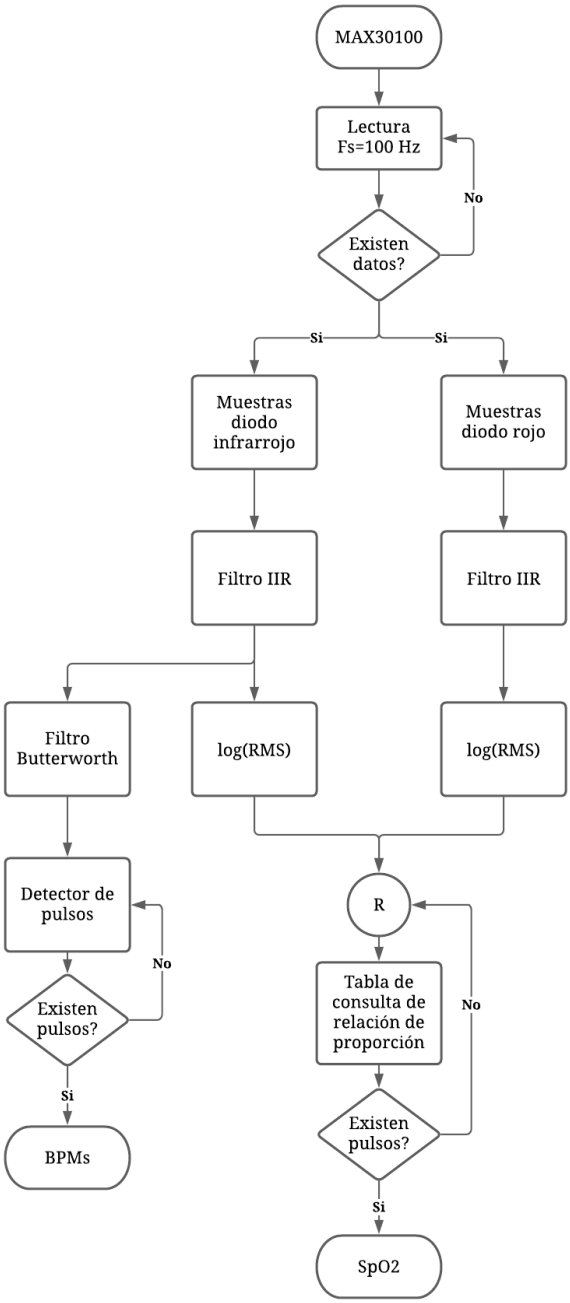


Fig. 40. Diagrama de flujo de funcionamiento de la librería MAX30100.

Elaborado por el investigador.

3.1.1.4.2. Programación de la placa Arduino Mini Pro

Para la programación de la placa es necesario realizar las conexiones de la pantalla OLED, el switch de encendido, el módulo bluetooth HC-05, las resistencias para la medición de la batería, el circuito de carga de la batería y el sensor MAX30100 como se muestra en la Fig. 41.

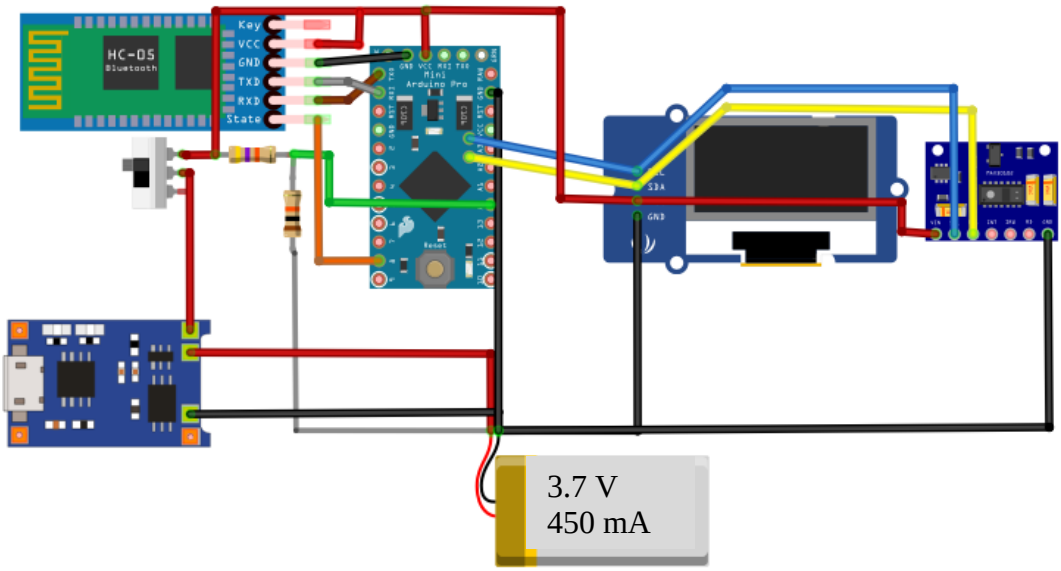


Fig. 41. Conexiones del circuito para la programación.

Elaborado por el investigador.

El circuito está conformado por bloques que ejecutan una función específica, las cuales se definen en la Fig. 42.

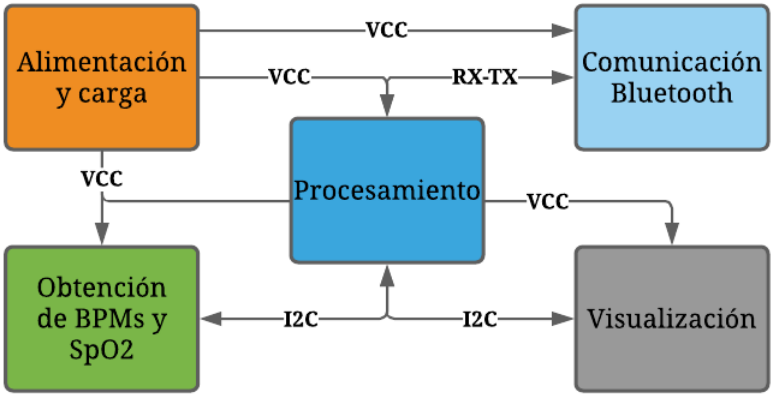


Fig. 42. Diagrama de bloques del circuito.

Elaborado por el investigador.

Alimentación y carga

Este bloque está compuesto por el circuito de carga de la batería y la batería. La batería de alimentación tiene un voltaje de 3.7 voltios a 450 mA, cuyos valores se han escogido según la tabla 19 donde se especifica el voltaje de funcionamiento y consumo de corriente de cada bloque del circuito. Estos valores se han obtenido de cada Datasheet del fabricante.

TABLA XVIII. Valores de consumo de corriente y voltaje de cada bloque según su fabricante. Elaborado por el investigador.

Bloque	Consumo de corriente (máximo).	Voltaje de alimentación
Comunicación Bluetooth	30 mA	3.3 V – 6 V
Procesamiento	150 mA	3.3 V – 12 V
Obtención de BPMs y SpO2	20 mA	3.3 V – 5 V
Visualización	12 mA	3.3 V – 5 V

Donde el total de consumo de corriente de todos los bloques es 212 mA. Dando un aproximado de dos horas de duración de la batería en constante funcionamiento. La medición del nivel de batería se realiza por el puerto A0, y se utilizan distintas variables para este proceso, como se ve en el siguiente código.

```
#define ANALOG 0
float outputValue = 0;
float smoothedVal = 0;
#define SAMPLES 15
#define VERF 4.5
```

Donde:

- ANALOG: es el puerto analógico del Arduino donde está conectada la batería.
- outputValue: es el valor medido desde el puerto analógico.
- smoothedVal: es el valor medido del puerto analógico pero suavizado.
- SAMPLES: es el número de muestras que se toman para realizar el suavizado de la señal medida desde el puerto analógico.

- VREF: es el voltaje de referencia con el que se realiza la medición.

Luego se realiza la medición y se suaviza la señal, devolviendo un rango de valores de 0.79 hasta 0, debido al arreglo de resistencias. Este arreglo se realiza para que la caída de tensión en el puerto no sea mayor, dificultando la alimentación de los otros bloques del circuito.

```
outputValue = (VERF * analogRead(ANALOG)) / 1024;
smoothedVal = smoothedVal + ((outputValue - smoothedVal) / SAMPLES);
```

Entonces se compara los distintos niveles de batería, donde se mostrarán imágenes en la pantalla Oled que indican el porcentaje de batería. Estos valores se calculan tomando en cuenta 4.1 V como valor máximo que puede tener la batería cargada, hasta 3.3 V que es el valor mínimo recomendado al que debe llegar la batería.

```
if (val > 0.79) {
  mostrargrafico(105, 0, carga, 19, 10);
}
if (val > 0.73 && val < 0.79) {
  mostrargrafico(105, 0, seis, 19, 10);
}
if (val > 0.66 && val < 0.73) {
  mostrargrafico(105, 0, tres, 19, 10);
}
if (val < 0.66) {
  mostrargrafico(105, 0, cero, 19, 10);
}
```

Estas imágenes se muestran gracias al método estático "*mostrargrafico*", el cual utiliza las variables de tipo "*static char*" guardadas en la memoria "Flash" del Arduino. Estos códigos se pueden revisar en el ANEXO L.

Comunicación Bluetooth

La comunicación Bluetooth se realiza mediante el módulo bluetooth HC-05. Este utiliza el estándar de comunicación IEEE 802.15.1 compatible con el IoT. Este funcionará a una tasa de transferencia de 9600 baudios y se utilizara el pin STATE para que el Arduino sepa cuando se ha realizado la vinculación con el sistema Android TV y poder mostrar datos útiles del usuario en la pantalla Oled. Este recibirá el nombre de usuario para mostrarlo en la Oled utilizando el siguiente código.

```
if (Serial.available() > 0) {
  incomingByte = Serial.read();
  if ((char)incomingByte == 'T') {
    memset(palabra, 0, sizeof(palabra));
    pSdata = palabra;
    oxi.c = 0;
  } else {
    *pSdata++ = (char)incomingByte;
  }
}
```

Siendo la variable palabra de tipo "*static char*" donde se almacena el texto.

```
static char palabra[15], *pSdata = palabra;
```

Luego se muestra en la pantalla Oled con la función estática "*mostrar*" en la posición 0 en X y 12 en Y.

```
mostrar(palabra, 0, 12);
```

A su vez, cuando el módulo esté enlazado y existan mediciones de BPMs enviará los datos separados por comas cada segundo. Incluido además un contador, el cual si llega a 60 indicará a la aplicación que la medición se ha realizado, y reiniciándose al mismo tiempo para futuras mediciones.

```
if (tiempoActual - tiempoAnterior >= intervaloEvento) {
  oxi.rate = pox.getHeartRate();
  oxi.spo = pox.getSpO2();
  if (oxi.rate && (char)incomingByte != 'T') {
    Serial.print(oxi.rate);
    Serial.print(',');
    Serial.print(oxi.spo);
    Serial.print(',');
    Serial.println(oxi.c);
    oxi.c++;
  }
  if (oxi.c == 60) {
    oxi.c = 0;
  }
  tiempoAnterior = tiempoActual;
}
```

Obtención de BPMs y SpO2

Estos datos se obtienen desde al sensor MAX30100 por medio de comunicación I2C. Se declara una variable a utilizar en todo el programa y que representa al sensor, que

en este caso se utilizará "*pox*". Además, se inicializa la librería al llamar al método "*pox.begin()*".

```
PulseOximeter pox;

pox.begin();
```

Luego se declaran una estructura que contienen las variables de medición de BPMs, SpO2, del contador y del estado del módulo HC-05. Este proceso permite definir un numero de bits a utilizar por cada variable, así se evita un colapso de la memoria del Arduino.

```
struct Oximeter {
  unsigned int rate: 8;
  unsigned int spo: 7;
  unsigned int c: 6;
  unsigned int estado: 1;
};

Oximeter oxi;
```

Luego se llama al método "*pox.update*()" continuamente en el "*loop*" del código de Arduino para que los datos obtenidos del sensor sean actualizados cada vez que un ciclo de código sea completado.

```
pox.update();
```

Luego se obtienen los datos de BPMs y SpO2 utilizando las funciones "*pox.getHeartRate()*" y "*pox.getSpO2()*" respectivamente cada segundo.

```
unsigned long tiempoActual = millis();
if (tiempoActual - tiempoAnterior >= intervaloEvento) {
  oxi.rate = pox.getHeartRate();
  oxi.spo = pox.getSpO2();
  tiempoAnterior = tiempoActual;
}
```

Se define 60 segundos como tiempo de medición ya que según la gráfica de la Fig. 43 los datos medidos por el sensor se estabilizan poco antes de este tiempo.

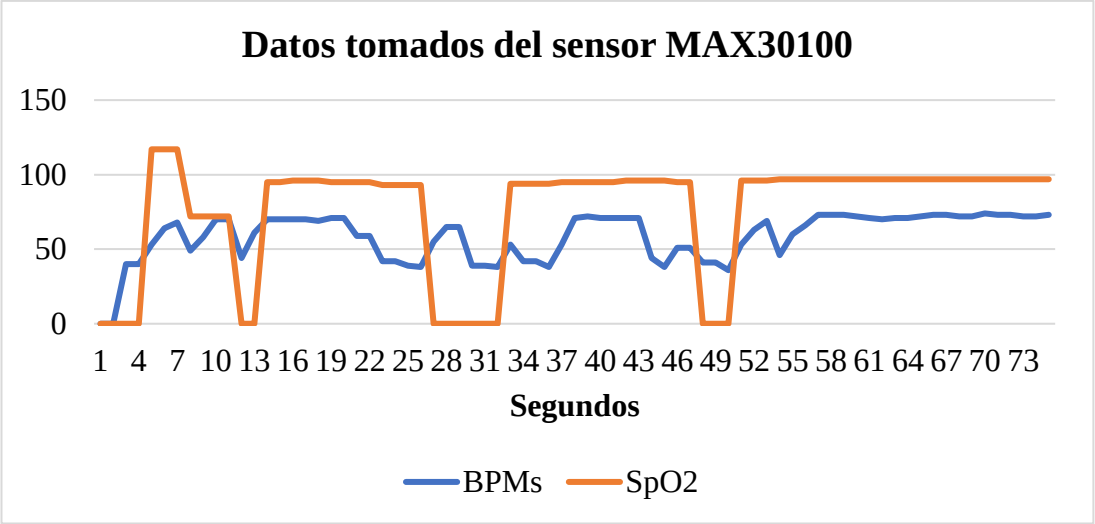


Fig. 43. Tiempo de estabilización de los datos medidos por el sensor MAX30100. Elaborado por el investigador.

Visualización

Esta parte está encargada de mostrar los datos de BPMs y SpO2, además del nombre de usuario que ha sido seleccionado desde la aplicación. Primero se declaran constantes de tipo "*char*" que almacenan texto en la memoria "Flash" del Arduino. Luego se crea una tabla que contienen estos textos, evitando sobrecargar la memoria RAM del Arduino.

```
const char BPMS[] PROGMEM = "BPMs";  //0
const char SPO[] PROGMEM = "SpO2";  //1
const char ESPERA[] PROGMEM = "Esperando conexion";  //2
const char ESCOJER[] PROGMEM = "Usuario:";   //3

const char *const string_table[] PROGMEM = {BPMS, SPO, ESPERA, ESCOJER};
char buffer[5];
```

Luego se crea un método de tipo "*char*" que devuelve el texto almacenado en la memoria "Flash" según la posición de la tabla que se le pida.

```
char* getStringfromProgMem(int i)
{
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));
  return buffer;
};
```

De esta forma se muestran los datos y sus etiquetas gracias al método estático "*mostrar*".

```
mostrar(oxi.rate, 15, 35);
mostrar(oxi.spo, 105, 35);
mostrar(getStringfromProgMem(0), 10, 52);
mostrar(getStringfromProgMem(1), 100, 52);
```

Cuando el módulo no esté enlazado, este mostrara el grafico llamado "*bluetooth*", caso contrario mostrara el grafico llamado "*cora*", utilizando el método estático "*mostrargrafico*".

```
mostrargrafico(42, 20, cora, 42, 42);

mostrargrafico(52, 22, bluetooth, 26, 42);
```

Estas dos imágenes están definidas como dos contantes de tipo "*unsigned char*" dentro de la memoria "Flash" del Arduino. El código fuente completo se puede encontrar en el Anexo J. El diagrama de flujo de funcionamiento del código de la placa Arduino Mini Pro se puede ver en la Fig. 44.

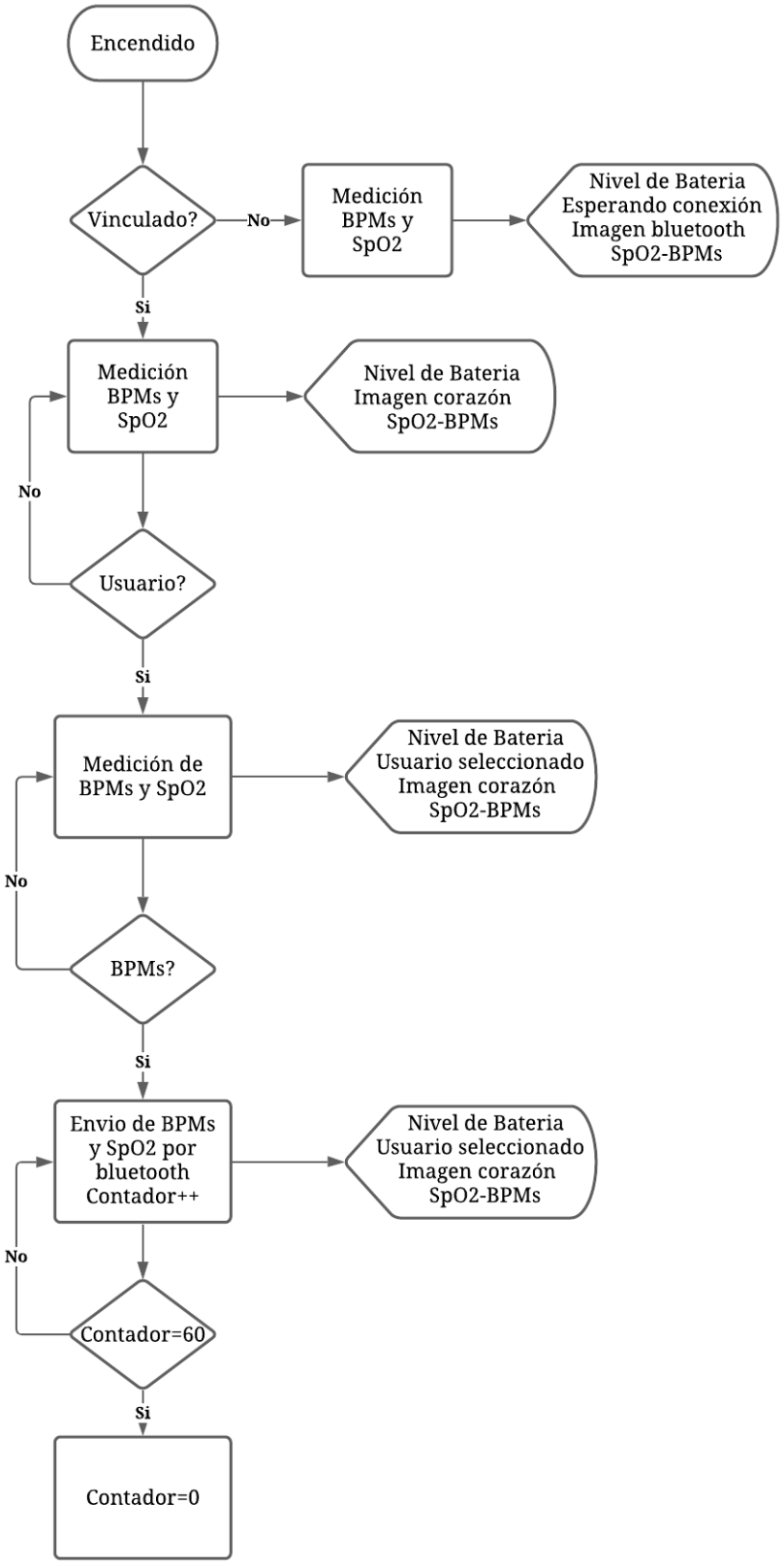


Fig. 44. Diagrama de flujo del código de la placa Arduino Mini Pro.

Elaborado por el investigador.

3.1.1.4.3. Diseño del circuito electrónico

El circuito electrónico unirá los diferentes elementos del circuito en un sistema embebido de medición y visualización de BPMs y SpO2. El diseño se lo realizará en el programa Proteus Profesional v8.9.

Conexión de los elementos

Para la conexión de los elementos que conforman el circuito electrónico se lo realiza en la sección de "Schematic Capture" del software Proteus. En la Fig. 45 se muestra las conexiones de los elementos del circuito previo al diseño del PCB (Placa de circuito impreso).

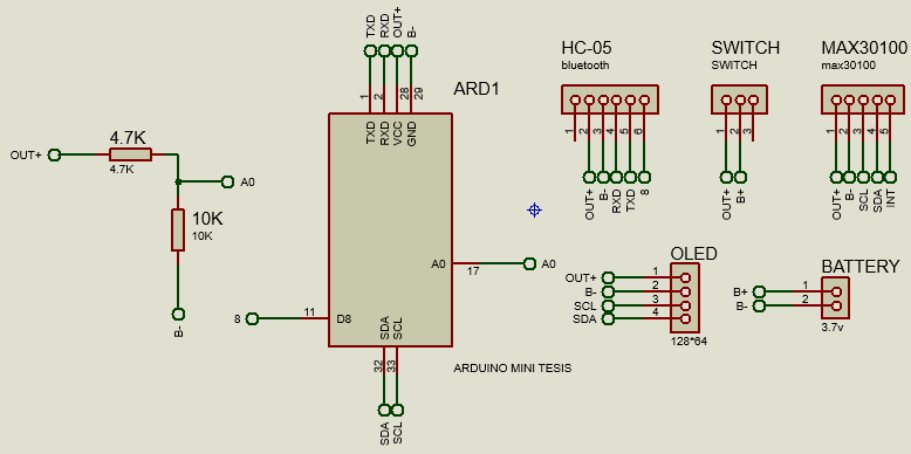


Fig. 45. Diagrama de conexión de los elementos del circuito electrónico hecho en Proteus Profesional v8.9.

Elaborado por el investigador.

Elaboración del PCB

La elaboración del PCB se la realiza en la sección "PCB Layout" de Proteus. Las características de diseño se definen en la tabla 20.

TABLA XIX. Características de diseño del PCB.

Elaborado por el investigador.

Característica	Valor
Ancho de pistas superiores	25 th (Thou) – 0.635 mm
Ancho de pistas inferiores	25 th (Thou) – 0.635 mm

Dimensiones agujeros de perforación para elementos	80 th diámetro externo x 30 th diámetro interno – 2.032 mm x 0.72 mm
Dimensiones de la placa	27.178 mm x 37.719 mm

El diseño final del PCB siguiendo los parámetros de la tabla 20 se muestra en la Fig. 46.

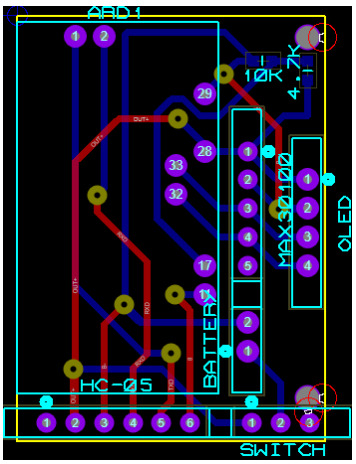


Fig. 46. Diseño PCB circuito electrónico.

Elaborado por el investigador.

3.1.1.5. Diseño de la aplicación para Android TV

Para la programación de la aplicación se utilizará el software Visual Studio Code, mediante el framework de programación React-Native. Antes de empezar a crear la aplicación es necesario tener instalado los siguientes requisitos de software (dependencias):

- NodeJS.
- Python.
- JDK de java.
- Android Studio.
- React-Native CLI.
- Visual Studio Code.
- React-Native Tools.

La instalación se puede revisar en el ANEXO M.

3.1.1.5.1. Creación de la aplicación

Para crear la aplicación se debe crear una carpeta la cual contendrá los archivos de esta. En esta carpeta se escribe el código "*npx react-native init Tesis*" en un terminal CMD de Windows como en la Fig. 47, donde Tesis es el nombre del proyecto.

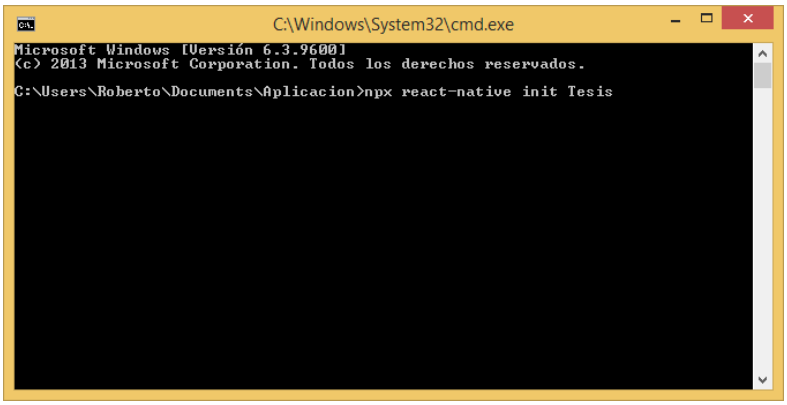


Fig. 47. Creación de la aplicación.

Elaborado por el investigador.

Una vez creada la aplicación los archivos se encontrarán en la dirección "*C:\Users\Roberto\Documents\Aplicacion\Tesis*" como se ve en la Fig. 48.

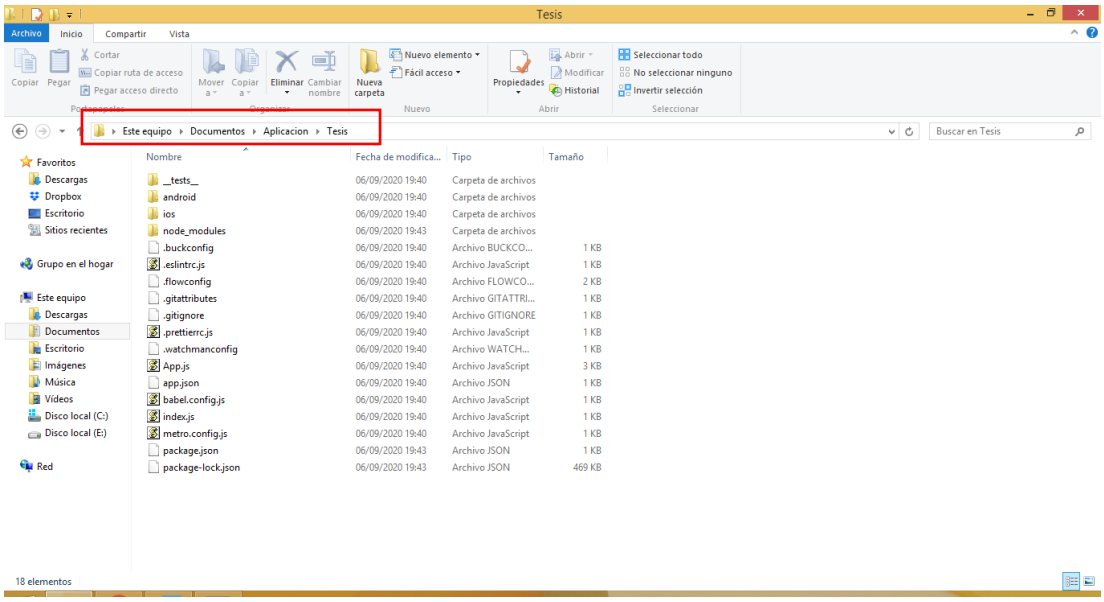


Fig. 48. Archivos de la aplicación.

Elaborado por el investigador.

NavigationContainer

Esta es responsable de la navegación entre las diferentes ventanas de la aplicación. Entre sus funcionalidades principales está la de actualizar la información o pasar información de los usuarios de una ventana a la otra. Para su instalación, se abre la carpeta del proyecto desde Visual Studio Code, abrir un terminal y escribir el código "*npm install @react-navigation/stack*" como en la Fig. 49.

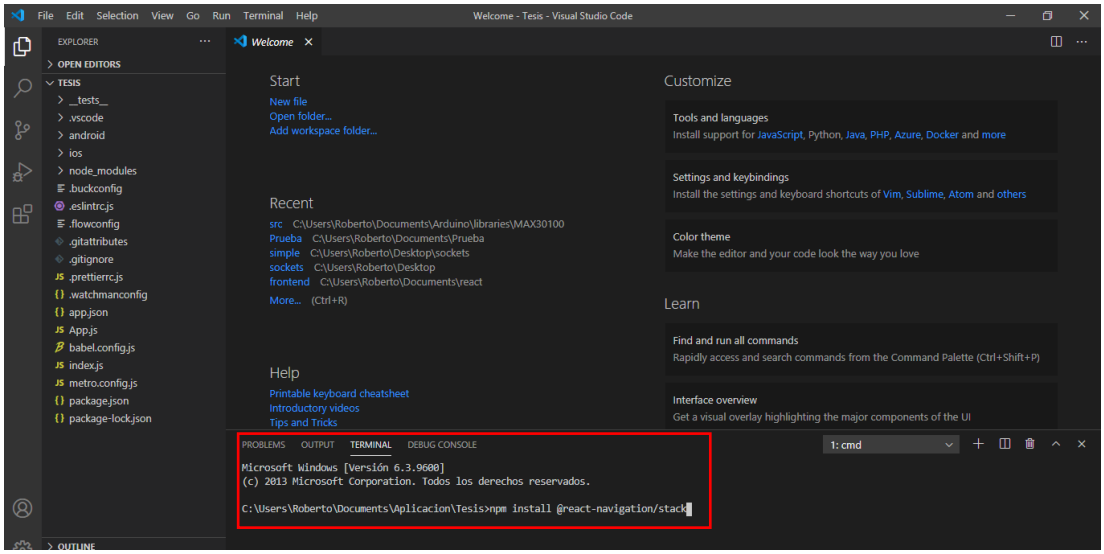


Fig. 49. Instalación de NavigationContainer.

Elaborado por el investigador.

Para la elaboración de la aplicación se seguirá el diagrama de la figura Fig. 50, donde se especifica el funcionamiento que tendrá y los datos que se mostrarán.

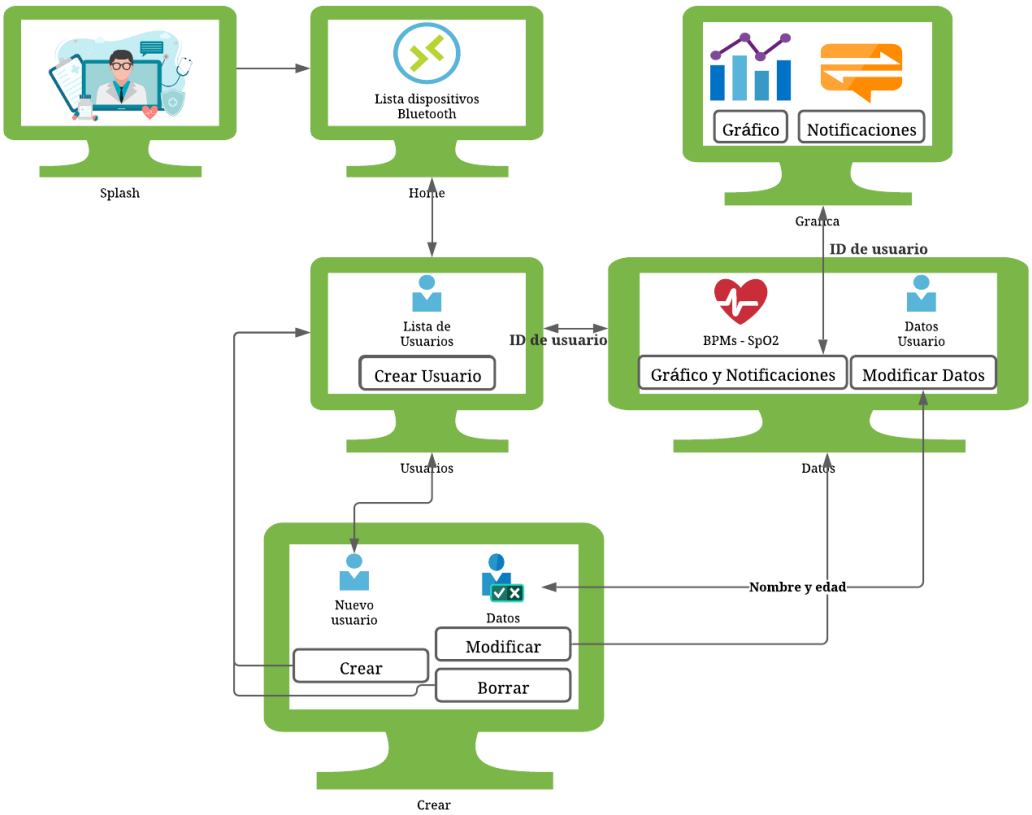


Fig. 50. Vistas de la aplicación.

Elaborado por el investigador.

Antes de comenzar con la programación, es necesario definir todos los comandos que se utilizarán en el proceso:

TABLA XX. Códigos para la implementación de la aplicación.

Elaborado por el investigador.

Comando	Función
async()	Una función "async" devuelve promesas que se resuelven con el valor de retorno de la función o se rechazan con errores no deseados.
await()	Es una función que espera a que se complete la promesa para continuar con la ejecución del siguiente código.
data	Es una variable de estado la cual almacenará los datos de la consulta realizada.
loading	Es una variable de estado que almacenará el estado de la consulta, si está ya se realizó o no, para continuar con la renderización de los demás componentes.
useState()	Es un "Hook" de React que se utilizará para asignar y actualizar los valores de las variables de estado.

useNavigation()	Es un “Hook” de React que permitirá utilizar el objeto que contiene los datos de los usuarios para pasarlo entre las ventanas y guardarlo en una variable de estado.
fetch()	Proporciona una respuesta o petición al realizar consultas web, generando un objeto de esta mediante promesas.
then()	Al iniciar con la petición fetch(), then() bloqueará la renderización de las vistas si los datos aún no han sido guardados en el objeto “data”. Este proceso se conoce como promesa.
setData()	Función que permitirá actualizar los valores dentro del objeto “data” cada vez que este es llamado.
setLoading()	Función que permitirá actualizar el valor de “loading” cada vez que se han consultado los datos.
useEffect()	Este “Hook” de React le dice al componente que debe ejecutar una función después de renderizarse, este llamará a la función “geData” para obtener los datos de la consulta.
addListener()	Esta monitorea los eventos que se generan en la aplicación. Esta trabajará con “focus”, la cual avisa que la vista actual ha sido llamada, llamando a la función “getData”. Esta es útil cuando se crea un nuevo usuario o se modifica su nombre.
map()	Esta función permite crear listas a partir de un array y guardarlo en un nuevo valor que será mostrado después en la vista. Esta mostrará los nombres de los usuarios en forma de lista dentro de botones en la página “Home” de la aplicación.
route.params	Es un objeto que guarda los valores enviados entre cada vista de la aplicación. Este es útil para saber que usuario se ha escogido para mostrar sus datos en la vista “Datos” o modificarlos en la vista “Crear”
JSON.stringify()	Transforma un objeto de tipo JavaScript en una cadena JSON. Esto es útil para cuando se quieran enviar peticiones al servidor donde se admite texto de tipo JSON.
navigate()	Es una “hélice de navegación” que permite indicar a cuál vista navegar cuando esta se llamada, definiendo el nombre de la vista dentro de la misma.
validateFields()	Esta función permitirá validar los datos ingresados al modificar o crear un usuario. Esta trabaja con la función “constraints”, donde se definen los caracteres permitidos y los mensajes a mostrar en caso de algún error.

Para la implementación de la aplicación se debe crear ocho archivos que alojaran el código de cada una de las ventanas a mostrar:

- App.js (en caso de que esta no sea creada por defecto).
- SplashScreen.js
- ListaBluetooth.js
- ListaUsuarios.js
- DatosPaciente.js
- Grafica.js
- CrearDatos.js
- Context.js

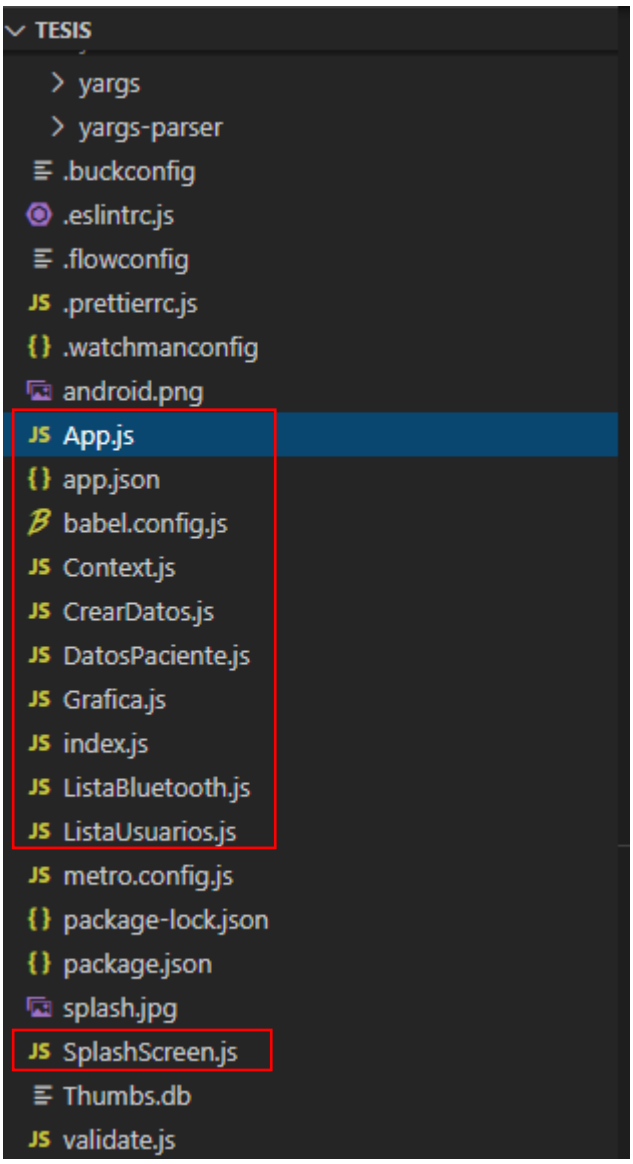


Fig. 51. Archivos para el código de las ventanas de la aplicación.

Elaborado por el investigador.

Cada archivo cumple una función específica dentro de la aplicación que a continuación se procede a explicar. Para la implementación de la comunicación Bluetooth y todos sus eventos se utilizará la librería "*react-native-bluetooth-serial-next*".

App.js

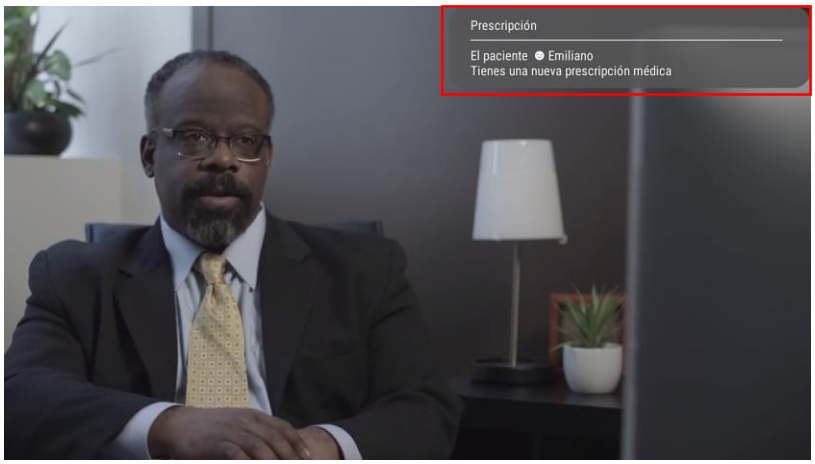
En este archivo se realizan tres funcionalidades diferentes:

- Se define el nombre de cada vista en función del componente que cada una renderiza, con el fin de llamarlas a su renderización cada vez que el usuario navega entre las vistas de la aplicación. Esto se define dentro del componente "*NavigationContainer*" el cual utilizará estos nombres dentro del código para navegar entre las vistas.
- Se realiza la conexión con el "*websocket*" implementado en Odoo para el recibir las notificaciones cuando una nueva notificación es enviada.

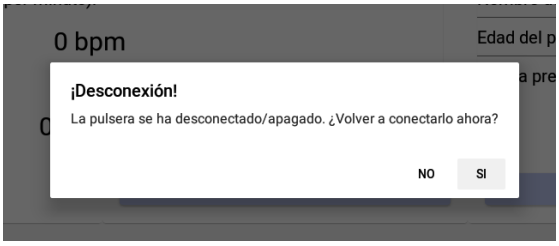
```
const ws = new WebSocket('ws://18.222.17.116:80', {
  transports: ['websocket'],
});

ws.onopen = () => {
  console.log('conectado');
};
ws.onerror = () => {
  console.log('error conectado');
};
```

Cuando se recibe un mensaje (notificación), este se decodifica de "*base64*" a "*ascii*" y se muestra mediante un "*Toast*" aun si la aplicación está en el modo "*background*" (en segundo plano).



- Se implementa un evento "*BluetoothSerial.on('connectionLost', ({device})"* de escucha cuando la pulsera es desconectada, apagada o se encuentra fuera del rango de comunicación. En caso de que la aplicación este en modo "*foreground*" (primer plano) se mostrará una advertencia preguntado al usuario si quiere realizar la conexión ahora o si la hará después. Si escoge hacerlo ahora esta navegará a la vista "*Home*" donde se realiza la vinculación bluetooth.



En caso de que la aplicación este en estado "*background*" se mostrará un "*Toast*" advirtiendo de la desconexión.

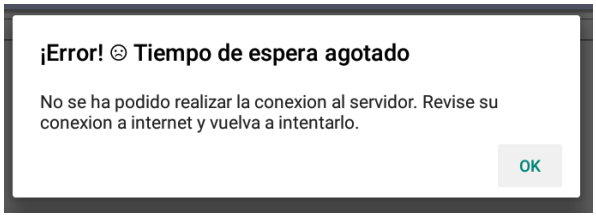
Debido al tamaño del código de "App.js" no se ha mostrado el código completo en esta sección, pero se puede revisar en el ANEXO N.

SplashScreen.js

Esta vista muestra una imagen de bienvenida mientras la API de Odoo "*/api/splash*" verifica si la ID única del dispositivo existe en la base de datos del modelo "*tesis.dispositivos*". Esta envía la ID mediante una petición "POST" a la dirección web "*http://18.222.17.116:8069/api/splash"* como un texto de tipo JSON. Al terminar este proceso la aplicación navegara a la pantalla principal donde se realiza la conexión bluetooth con el circuito de la pulsera.

```
const values = {
  codigo: DeviceInfo.getUniqueId(),
};
await fetch('http://18.222.17.116:8069/api/splash', {
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  method: 'POST',
  body: JSON.stringify(values),
})
```

En caso de existir algún error debido a la interrupción de la conexión a internet o con el servicio de Odoo, se mostrará una alerta advirtiendo al usuario que no se ha podido realizar la petición. Se deberá reiniciar la aplicación para volver a intentar este proceso.



La vista generada por este archivo se muestra en la Fig. 52. El código completo de este archivo se puede encontrar en la ANEXO O.

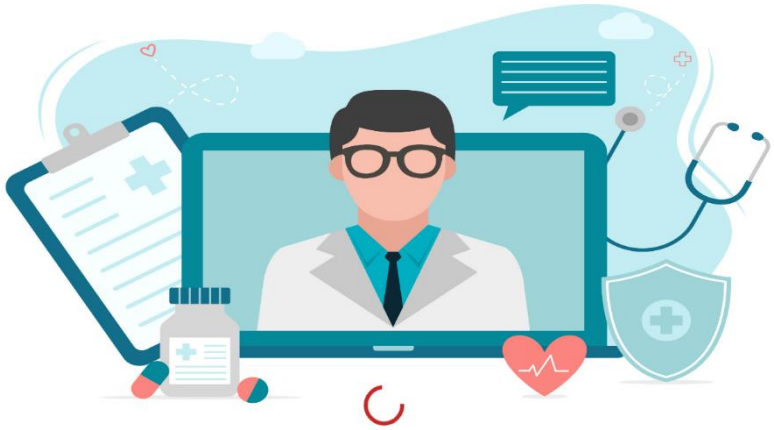


Fig. 52. Vista de bienvenida a la aplicación.

Elaborado por el investigador.

ListaBluetooth.js

Esta vista permitirá al usuario vincular el sistema Android TV con la pulsera por medio de Bluetooth. Al navegar desde la vista "Splash" la aplicación mostrará una imagen indicando que el bluetooth esta apagado junto a un "*ToggleSwitch*" para habilitar la

conexión Bluetooth en caso de que no lo esté, y un botón con el texto “Continuar sin conectar la pulsera” en caso de que el usuario no desee conectar la pulsera, navegando a la vista “*Usuarios*”.

Fig. 53. Vista “Bluetooth” antes de activar las conexiones Bluetooth.
Elaborado por el investigador.

Una vez que se activa el bluetooth se desplegará una lista con todos los dispositivos vinculados con el sistema Android TV.

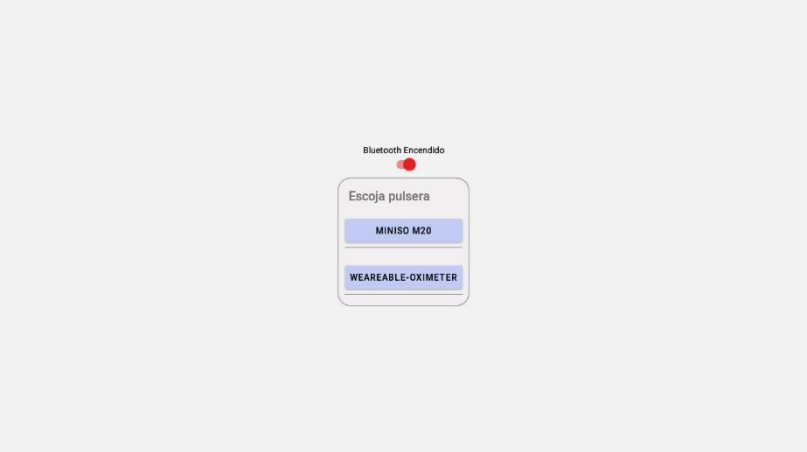


Fig. 54. Vista "Bluetooth" una vez activada la conexión Bluetooth.
Elaborado por el investigador.

En esta lista se realiza una comprobación para saber si el nombre del dispositivo es “*Weareable-Oximeter*” y realizar la conexión, caso contrario se mostrará un “*Toast*” indicando que no es el dispositivo correcto.

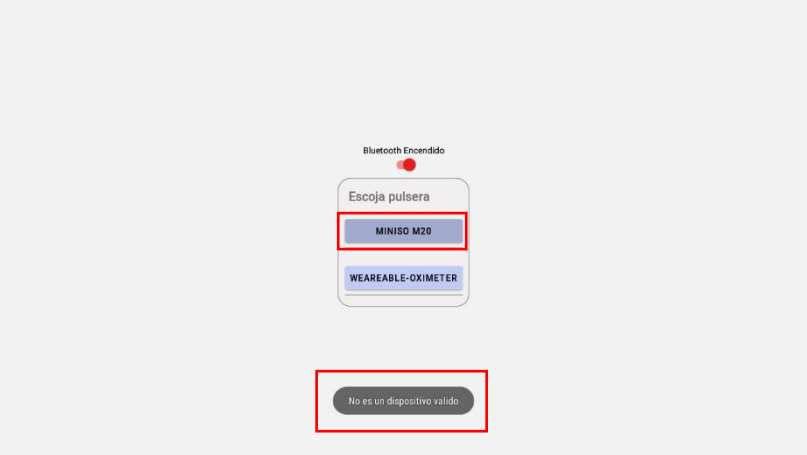


Fig. 55. Error al seleccionar un dispositivo a vincular diferente a la pulsera.

Elaborado por el investigador.

Caso contrario se realiza la conexión y se despliega la lista de usuarios. En el caso de que la pulsera no esté encendida u ocurrió un problema en la conexión, la aplicación mostrar un mensaje advirtiendo al usuario que ha ocurrido un error.

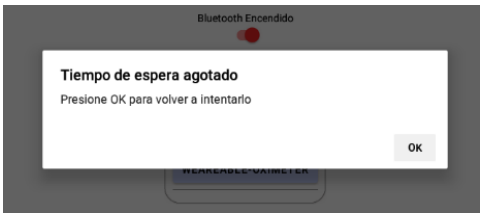


Fig. 56. Error al vincular la pulsera.

Elaborado por el investigador.

El código puede ser revisado en el ANEXO P.

ListaUsuarios.js

Los usuarios se mostrarán en una lista cuyos nombres se obtienen al realizar una petición "*fetch*" desde la dirección web "*http://18.222.17.116:8069/api/datos/IDdispositivo*", mediante el método "GET". Se guardan todos los valores en el objeto "*data*" al llamar a la función "*setData*" cada vez que se realiza la consulta. Se le pasa el ID único del dispositivo para mostrar solo los usuarios que pertenecen al dispositivo que realiza la consulta.

```
await fetch(
  'http://192.168.1.11:8069/api/datos/' + DeviceInfo.getUniqueId(),
  {
    headers: {
      Accept: 'application/json',
    },
    signal: signal,
    method: 'GET',
  },
)
```

En caso de existir un error en la consulta, se mostrará el mismo mensaje de advertencia que la vista "*Splash*". Se debe volver a navegar en la vista para volver a intentarlo. Después se sacan todos los valores que tengan el valor "*nombre*" y se muestra en la lista con la función "*map*" dentro del componente "*Mapeo*" con un encabezado "*Seleccione usuario*", dentro de botones que servirán para la selección y envió de los datos correspondientes de cada usuario a la vista "*Datos*".

```
const Mapeo = () => {
  return data.map((val, index) => (
    <View key={index} style={{padding: 10}}>
      <Button
        color="#C1CBF4"
        compact={true}
        mode="contained"
        onPress={() => navigate.navigate('Datos', val)}>
        <Text>{val.nombre}</Text>
      </Button>
      <Separator />
    </View>
  ));
};
```

Además, renderiza un botón para la creación de un nuevo usuario, que no enviará la "*id*" ya que no se modificarán los datos de algún usuario. El código completo de "*ListaUsuario.js*" se puede encontrar en el ANEXO Q. Esta vista se muestra en la Fig. 57.

Fig. 57. Vista "Usuarios" de la aplicación.

Realizado por el investigador.

DatosPaciente.js

Este archivo realiza dos funciones principales:

- Mostrar los datos del paciente y botones: una vez pasados los datos de la vista "*Usuarios*" estos se guardan en el objeto "*data*". Estos se pasan desde la vista anterior gracias al objeto "*route.params*" que utiliza la dependencia de navegación.

```
const [data, setData] = useState({
  id: route.params.id,
  nombre: '',
  edad: '',
});
```

  Estos datos se muestran en un componente de tipo Card a la derecha de la pantalla. Ademas de un botón que permite modificarlos.

```
<Card>
  <Card.Title
    title="Datos"
    subtitle="Pulse el boton 'Editar Datos' para editar los datos"
    left={User}
  />
  <Text style={{fontSize: 20}}>
    Nombre del paciente: {data.nombre}
  </Text>
  <Separator />
  <Text style={{fontSize: 20}}>Edad del paciente: {data.edad}</Text>
  <Separator />
  <Text style={{fontSize: 20}}>Última prescripción: {mensaje}</Text>
</Card>
```

Consulta la ultima prescripcion enviada desde la API "*/api/prescripcion/id*" con el "*id*" del usuario que se ha pasado en la navegacion.

```
await fetch(
  'http://18.222.17.116:8069/api/prescripcion/' + route.params.id,
  {
    headers: {
      Accept: 'application/json',
    },
    signal: signal,
    method: 'GET',
  },
)
```

Realiza la consulta de los datos del paciente desde la API "*/api/paciente/id*" en caso de que estos se hayan modificado mediante el metodo "*fetch*".

```
await fetch('http://18.222.17.116:8069/api/paciente/' + route.params.id, {
  headers: {
    Accept: 'application/json',
  },
  signal: signal,
  method: 'GET',
})
```

Muestra un botón para navegar hacia la vista "*Grafico*" pasando el "*id*" del usuario para realizar la consulta de los datos en funcion de esta.

```
<Button
  compact={true}
  color="#C1CBF4"
  mode="contained"
  onPress={() => navigate.navigate('Grafica', data.id)}>
  <Text>Grafico y Notificaciones</Text>
</Button>
```

- Comunicación bluetooth y subida de datos: a la izquierda de la vista se muestran los datos que son recibidos desde la pulsera. Estos se almacenan en la variable "signos" mediante el método "setSignos". Además del contador que

el Arduino realiza en su programación. La medición se interrumpirá si el usuario navega a otra ventana.

```
const [signos, setSignos] = useState({
  presion: 0,
  oxigeno: 0,
  contador: 0,
});
```

Esta enviara el nombre de usuario mediante el método "*BluetoothSerial.writeToDevice*" de la librería "*react-native-bluetooth-serial-next*" de React Native, cada vez que se navega dentro de esta vista.

```
useEffect(() => {
  write(idDevice, route.params.nombre);
}, [navigate]);
```

Estos datos se reciben tanto si la aplicación está en segundo plano o no. Para ello se implementa un "*BackgroundService*".

```
const TareaBack = async () => {
  try {
    await BackgroundService.start(read, options);
  } catch (e) {
    console.log('Error', e);
  }
};
```

Al cual se le pasa el método "*read()*" donde se ejecuta la lectura de datos enviados por bluetooth mediante el método "*BluetoothSerial.readEvery*" de la librería "*react-native-bluetooth-serial-next*" de React Native. El cual separa los datos por medio de una coma enviada desde el código de Arduino.

```
BluetoothSerial.readEvery(
  (data, intervalId) => {
    console.log(data);
    var arrayDeCadenas = data.split(',');
    for (var i = 0; i < arrayDeCadenas.length; i++) {
      arrayDeCadenas[i];
    }
    if (arrayDeCadenas[1] == undefined) {
      setSignos({
        ...signos,
        presion: 0,
        oxigeno: 0,
      });
    } else {
      setSignos({
        ...signos,
        presion: arrayDeCadenas[0],
        oxigeno: arrayDeCadenas[1],
        contador: arrayDeCadenas[2],
      });
    }
```

Cuando el valor del contador es igual a 60 se mostrará una alerta que indica la terminación de la medición, cuando la aplicación está en modo "*foreground*".

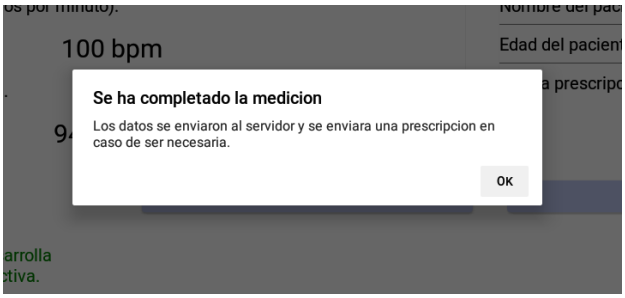


Fig. 58. Alerta de indicación de los datos subidos cuando la aplicación está en modo "foreground".

Realizado por el investigador.

En caso de que este en modo "*background*" se mostrara un Toast con los valores medidos.

Fig. 59. Mensaje Toast una vez terminada la medición cuando la aplicación está en modo "background".

Realizado por el investigador.

Enseguida subirá los datos mediante la "*/api/datosPaciente*"

Renderiza un texto que permite saber si la pulsera está conectada o no.

```
<Text style={{color: !conected ? 'red' : 'blue'}}>{`${
  !conected ? 'PULSERA DESCONECTADA' : 'PULSERA CONECTADA'
}`}</Text>
```

Esta función se implementa de manera global para identificar si la pulsera se ha desconectado. El archivo "Context.js" permite almacenar variables globales que pueden ser utilizadas por cualquier vista de la aplicación. Esta se importa en la vista en la que se quiere utilizar y se importan los métodos y variables a manejar, en este caso se usara la variable "*conected*" que permite saber si la pulsera se ha desconectado, e "*idDevice*" para escribir por bluetooth el nombre del usuario. Estos datos ya se han escrito en la vista "Home".

```
import {useAppContext} from './Context';

  const {idDevice} = useAppContext();
  const {conected} = useAppContext();
```

El funcionamiento del archivo "Context.js" se explica posteriormente en este capítulo.

El código fuente completo de "DatosPaciente.js" puede ser encontrado en el ANEXO R. La vista se muestra en la Fig. 60.

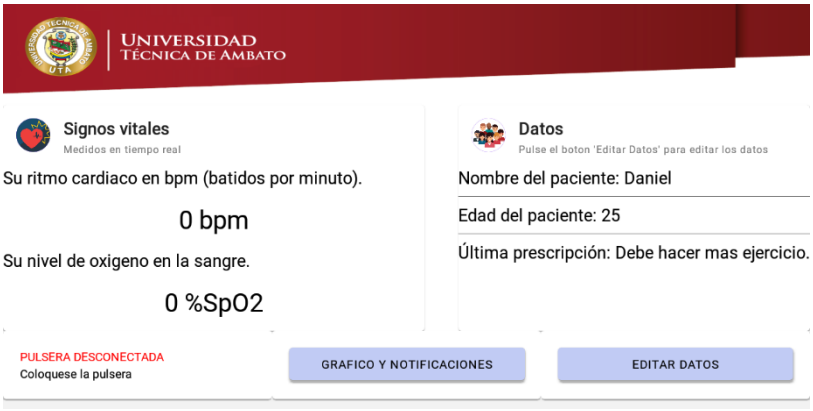


Fig. 60. Vista "Datos" de la aplicación.

Realizado por el investigador.

Crear.js

Una vez pasados los datos del usuario para ser modificados, se condiciona la vista en caso de que exista un "*id*" que se haya pasado mediante el objeto "*route.params*" y se los guarda en el objeto "*datos*" llamando a "*setDatos*". Caso contrario, el objeto "*datos*" quedara vacío para almacenar los datos del nuevo usuario.

```
if (route.params.id) {
  setDatos({
    edad: route.params.edad,
    edadError: '',
    nombre: route.params.nombre,
    nombreError: '',
    email: route.params.email,
    emailError: '',
    doctor: route.params.medico,
    medicoError: '',
    boton: 'Modificar',
    mensaje: 'Usuario Modificado',
    borrar: true,
  });
}
```

Se condiciona la existencia del "*id*" para seleccionar entre las APIs de modificación o creación, y los métodos "PUT" y "POST", mediante la dirección web "*http://18.222.17.116:8069/api/modificar_usuario*" y "*http://18.222.17.116:8069/api/crear_usuario*", mediante la función "*JSON.stringify*" se envía la variable "*values*" que contiene los datos ingresados en los "*TextInput*".

```
await fetch(
  !route.params.id
    ? 'http://18.222.17.116:8069/api/crear_usuario'
    : 'http://18.222.17.116:8069/api/modificar_usuario',
  {
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    method: !route.params.id ? 'POST' : 'PUT',
    body: JSON.stringify(values),
  },
)
```

Si se han pasado datos, estos se renderizan en un "*TextInput*", caso contrario estos estarán vacíos para un nuevo usuario. Si existe una "*id*" se mostrarán dos botones, uno para "*Modificar*" y otro para "*Borrar*". El botón "*Modificar*" es el mismo del anterior "*Crear*" solo que su mensaje cambia si existe un "*id*". Este mensaje se almacena en el objeto "*datos.boton*".

```
<Button
  compact={true}
  color="#5AADF1"
  mode="contained"
  onPress={crearPaciente}>
  <Text>{datos.boton}</Text>
</Button>
```

Al modificar el usuario, la aplicación navegará a la vista "*Datos*", y al crear se navegará a la vista "*Usuarios*".

```
.then((response) => response.json())
.then((responseJson) => {
  !route.params.id
    ? navigation.navigate('Usuarios')
    : navigation.navigate('Datos');
})
```

El otro servirá para borrar el usuario mediante la dirección web "*http://18.222.17.116:8069/api/borrar*".

```
await fetch('http://192.168.1.11:8069/api/borrar', {
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  method: 'POST',
  body: JSON.stringify({id: route.params.id}),
})
  .then((response) => response.json())
  .then((responseJson) => {
    navigation.navigate('Usuarios');
  })
```

Antes de borrar el usuario se mostrará un mensaje de verificación, y si se confirma al presionar el botón "*Aceptar*" este llamará a la función "*borrarUsuario*" del código anterior y se enviará a borrar mediante el "*id*" antes pasado desde la vista "*Datos*", navegando a la vista "*Usuarios*" al terminar este proceso.

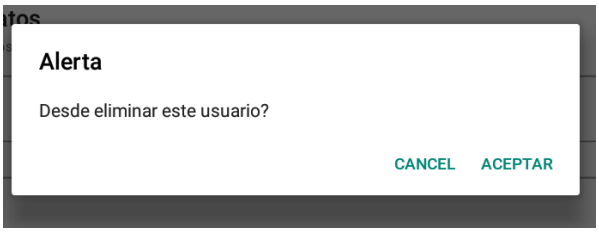


Fig. 61. Alerta de confirmación antes de eliminar un usuario.

Realizado por el investigador.

El código fuente completo de "*CrearDatos.js*" se puede encontrar en el ANEXO S. La vista "Crear" se muestra en la Fig. 62.

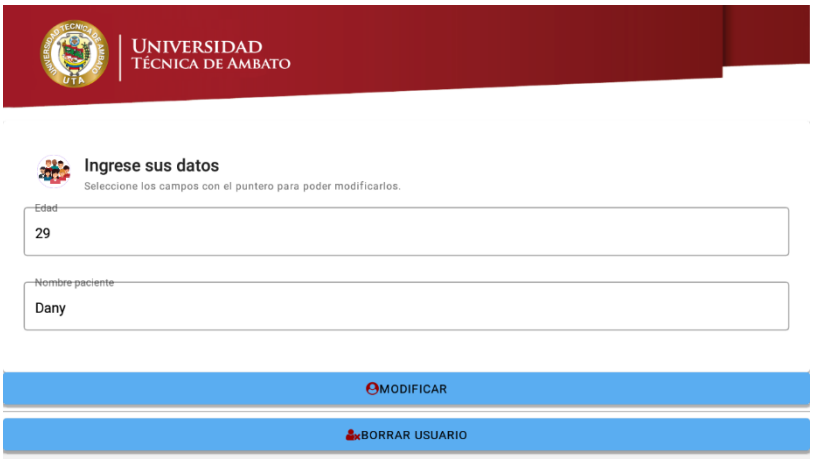


Fig. 62. Vista "Crear" de la aplicación.

Realizado por el investigador.

Grafica.js

Este archivo permite filtrar los datos de la gráfica y las notificaciones por fechas, además de renderizar la vista "*Grafica*". El filtrado de datos se logra al realizar una petición a la dirección web "*http://192.168.1.11:8069/api/grafica*" al enviar las fechas en texto JSON desde la aplicación. La librería por usar en este caso se llama "*moment*", la cual formatea las fechas para que la API de Odoo las pueda procesar.

```
await fetch('http://192.168.1.11:8069/api/grafica', {
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  method: 'POST',
  body: JSON.stringify({
    res_paciente: route.params,
    start_date: new Date(
      moment(fechaInicio).set({h: 0, m: 0}),
    ).toISOString(),
    end_date: new Date(
      moment(fechaFinal).set({h: 23, m: 59}),
    ).toISOString(),
  }),
})
```

Estas fechas se definen al llamar a los métodos "*setInicio*" para la fecha desde la cual se quiere filtrar, y "*setFinal*" para la fecha final. En el caso de la gráfica se utilizará la librería "*react-native-chart-kit*", la cual necesita un objeto con los valores a mostrar. Este objeto está construido por los siguientes valores:

- **Labels:** estos valores representan a los mostrados en el eje Y de la gráfica, aquí se mostrarán las fechas en las que se han subido los datos.
- **Datasets:** es un objeto el cual contiene el array "*data*" que son los valores que renderizar en la gráfica, además se define el "*id*" y el "*color*" en el cual serán renderizados los valores. Este puede tener uno o varios objetos con arrays "*data*", "*id*" y "*color*" diferentes.
- **Legend:** este es un array que define el nombre que tendrá cada curva que se muestra en la gráfica.

El objeto que es enviado desde la API se muestra a continuación:

```
{
  'data': {
    'datasets': [{'id': 1, 'data': []}, {'id': 2, 'data': []}],
    'labels': label,
  },
  'notificacion': {
    'datos': notis, 'contador': contador
  }
}
```

En los arrays "*data*" la API añade todos los valores en el rango de fechas que se ha especificado en la petición, al igual que "*labels*". En el objeto "*datos*" se añaden todas las notificaciones en ese mismo rango, además del "*contador*" que cuenta todas las notificaciones que aún no han sido leídas para mostrar en la aplicación. Para la gráfica este objeto se descompone por código de la siguiente forma.

```
setDatos({
  labels: responseJson.result.data.labels,
  datasets: [
    {
      id: responseJson.result.data.datasets[0].id,
      data: responseJson.result.data.datasets[0].data,
      color: (opacity = 1) => `rgba(17, 112, 201, ${opacity})`,
    },
    {
      id: responseJson.result.data.datasets[1].id,
      data: responseJson.result.data.datasets[1].data,
      color: (opacity = 1) => `rgba(201, 17, 17 ,  ${opacity})`,
    },
  ],
  legend: ['SpO2','BPMs'], // optional
});
```

Donde el método "*setDatos*" guarda el objeto recibido de la consulta en la variable "*datos*", el cual se utiliza en el método de renderizado de los valores en la gráfica. El objeto "*responseJson*" representa el objeto que se ha recibido al realizar la consulta. Para las notificaciones, se usa el mismo objeto para almacenarlo en una variable "*noti*" mediante el método "*setNoti*".

```
setNoti(responseJson.result.notificacion);
```

Luego estas son renderizadas en forma de botones utilizando la función "*map*". Estas luego se insertarán en un componente "*ScrollView*" para que el usuario pueda interactuar fácilmente.

```
const MapeoNotis = () => {
  return noti.datos.map((val, index) => {
    return (
      <View key={index}>
        <Button style={{nextFocusDown: true}}>
          <Text style={{fontSize: 20}}>
            {val.fecha} {val.mensaje}
          </Text>
        </Button>
      </View>
    );
  });
};
```

Una vez que el usuario navegue en esta vista y vea las notificaciones, la aplicación enviara el "*id*" del usuario a la API "*http://192.168.1.11:8069/api/actNoti*" para que se actualice el estado de las notificaciones a leídas, evitando que el contador de la API anterior las cuente como no leídas y mostrar al usuario solamente las que aún no han sido vistas.

```
const values = {
  res_paciente: route.params,
};
await fetch('http://192.168.1.11:8069/api/actNoti', {
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  method: 'PUT',
  body: JSON.stringify(values),
})
```

Debido al tamaño del código no se ha mostrado completamente en esta sección por lo que puede ser revisado en el ANEXO T al final de este documento. Para finalizar se insertan botones para que el usuario pueda seleccionar las fechas, para buscar renderizar la gráfica o la vista de las notificaciones. Estas vistas se muestran en la Fig. 63.

Fig. 63. Vista "Grafica" de la aplicación.

Realizado por el investigador.

Context.js

En este archivo proporcionan métodos y variables de estado globales que pueden ser utilizadas en todas las vistas de la aplicación. Esta función permite guardar el "*id*" de la pulsera mediante la variable "*idDevice*" mediante el método "*setDevice*". Esta variable permite realizar las siguientes funcionalidades:

- Cada vez que el usuario navega dentro de la vista "*Datos*" el nombre del usuario es enviado por bluetooth, y cuando se navega a otra vista, se limpia la pantalla Oled del circuito al enviar un dato que el Arduino entiende como el comando de limpieza.
- Para limpiar el buffer de las mediciones por bluetooth cada vez que se navegue fuera de la vista "*Datos*", ya que los valores anteriores pueden interferir en la toma real de los datos enviados desde la pulsera.
- Detener la medición de los datos cuando se navega fuera de la vista "Datos". Esto es vital para que el proceso de limpieza del buffer, ya que ocurriría un error si se quiere realizar la limpieza mientras el método "*read*" siga operando.
- Para renderizar en la vista "*Datos*" un mensaje advirtiendo si la pulsera está conectada con el sistema Android TV. Cada vez que se conecta un dispositivo en la vista "*ListaBluetooth*" se utiliza el método "*isConected*" para cambiar a la variable de estado "*conected*" a "*true*". En el caso de que la pulsera se desconecte, se usa el mismo método para cambiar la variable a "*false*", sin importar en que vista se encuentre el usuario.

Como se muestra en el ANEXO N, en el archivo "*App.js*" se implementa el componente "*AppProvider*" en la función "*App*", lo que indica que el "*Context*" se usa en todas las vistas de la aplicación. El código implementado se muestra a continuación.

```
import React, {useContext, useState, createContext} from 'react';

const AppContext = createContext({
  idDevice: '',
  setIdDevice: () => {},
  conected: false,
  isConected: () => {},
});

export const AppProvider = (props) => {
  const [idDevice, setIdDevice] = useState('');
  const [conected, isConected] = useState(false);

  return (
    <AppContext.Provider
      value={{
        idDevice,
        setIdDevice,
        conected,
        isConected,
      }}>
      {props.children}
    </AppContext.Provider>
  );
};

export const useAppContext = () => useContext(AppContext);
```

El funcionamiento de la aplicación se explica en el diagrama UML de caso de uso de la Fig. 64.

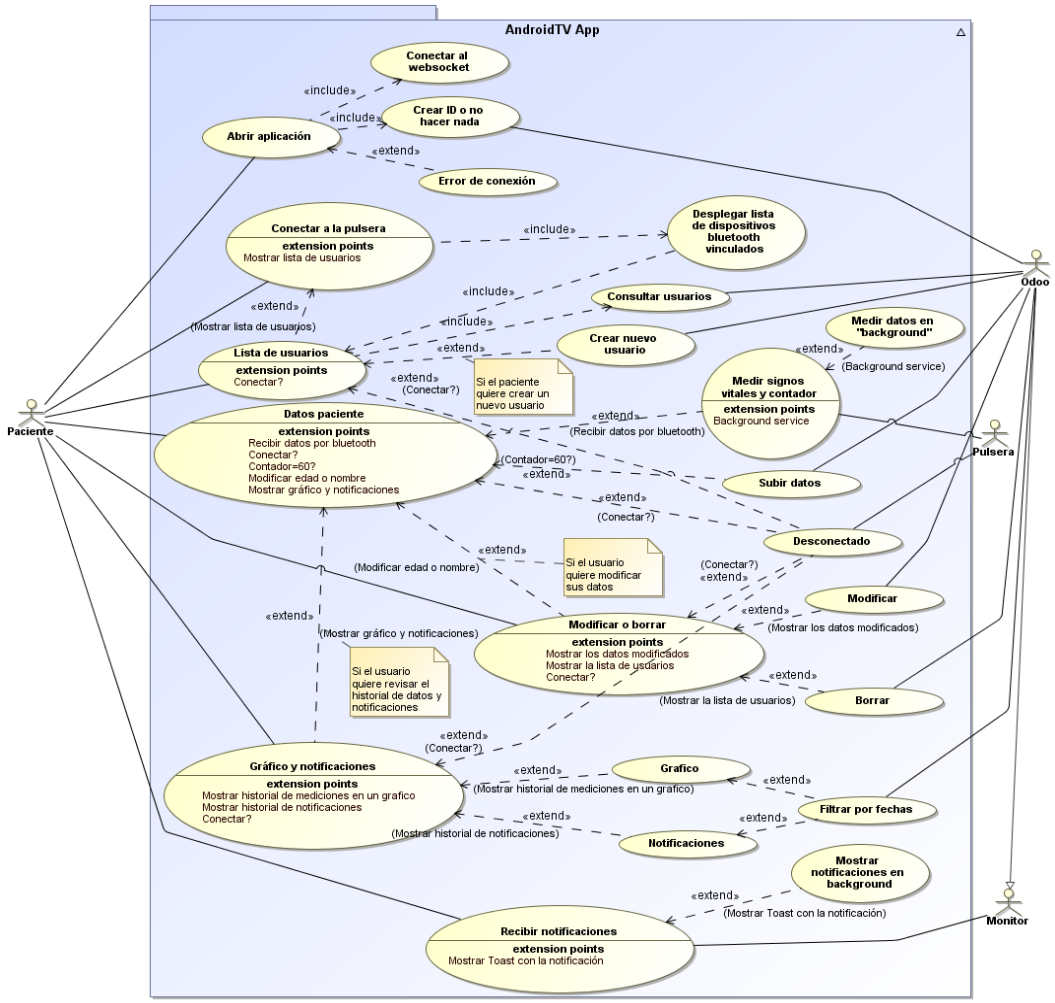
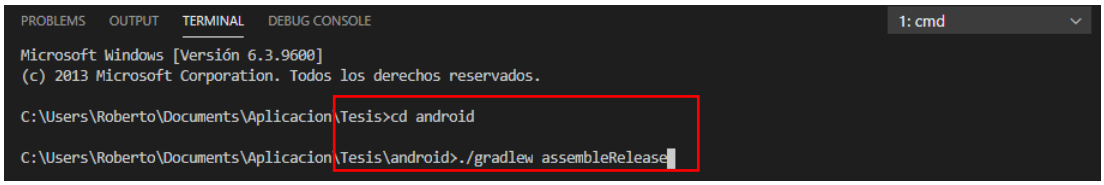


Fig. 64. Diagrama UML de caso de uso de la aplicación para Android TV. Realizado por el investigador.

3.1.1.5.2. Producir el APK de la aplicación

Para realizar este proceso se debe escribir el comando "*./gradlew assembleRelease*" en el terminal de Visual Studio Code donde, en la capeta "*android*" donde se encuentra el proyecto.



Luego el archivo estará ubicado en la dirección "*C:\Users\Usuario\Documents\Aplicacion\Tesis\android\app\build\outputs\apk*"

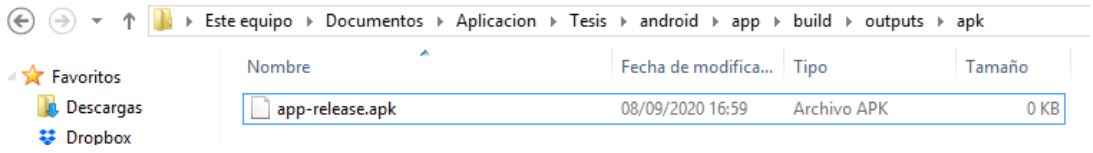


Fig. 65. Aplicación generada.

Realizado por el investigador.

Luego se la debe instalar por medio de USB en la Tv-box

Fig. 66. Instalación de la aplicación en la Tv-box.

Realizado por el investigador.

3.1.1.6. Diseño 3D de la pulsera

El diseño de la pulsera se llevará a cabo en el software "*Solid Works 2017*". Este permitirá generar un archivo de tipo STL (estereolitografía) para impresión en 3D. Se utilizaron las diferentes herramientas de SolidWorks para su diseño. En la Fig. 67 se muestran las dimensiones de la pulsera.

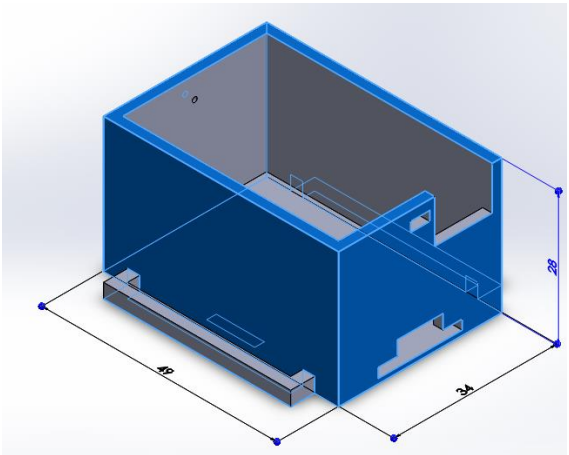


Fig. 67. Dimensiones de la pulsera.

Realizado por el investigador.

Estas dimensiones se obtuvieron al medir el tamaño del circuito ya implementado. En la parte interior posee una abertura para el circuito de carga y un agujero para los cables de conexión del sensor.

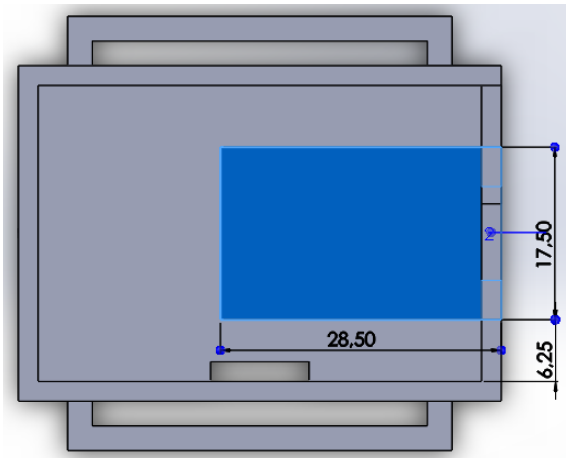


Fig. 68. Vista superior del diseño de la pulsera.

Realizado por el investigador.

Al costado derecho la pulsera tiene diversas aberturas.

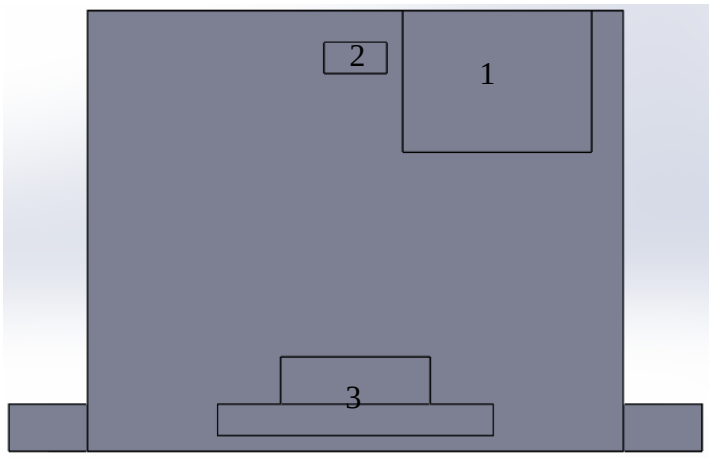


Fig. 69. Vista lateral derecha del diseño de la pulsera.

Realizado por el investigador.

Donde:

1. Abertura para el switch de encendido del circuito.
2. Abertura para la sujeción de la tapa de la pulsera.
3. Abertura para la conexión del circuito de carga.

En el otro costado, posee un agujero para la sujeción de la tapa del circuito por medio de un tornillo.

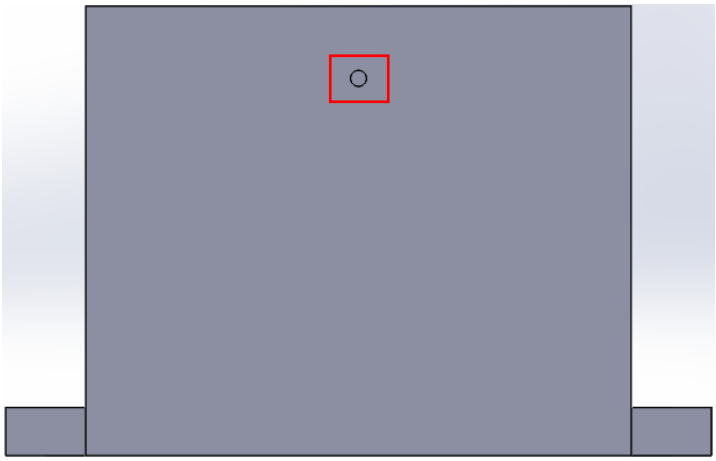


Fig. 70. Vista lateral izquierda del diseño de la pulsera.
Realizado por el investigador.

También posee bases para la sujeción de una correa que ayudará a su colocación en la muñeca del usuario.

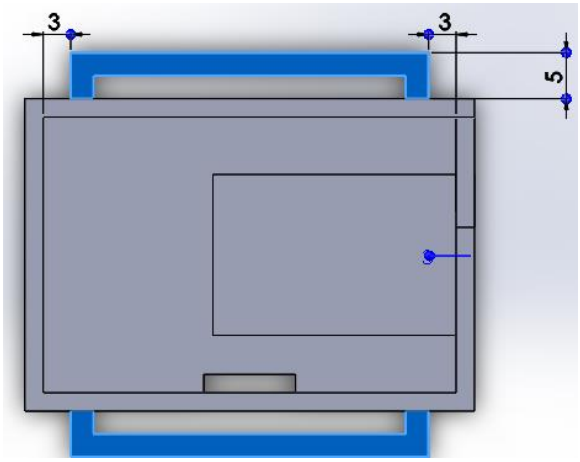


Fig. 71. Bases para la sujeción de la correa.
Realizado por el investigador.

La tapa de la pulsera posee las dimensiones de la Fig. 72.

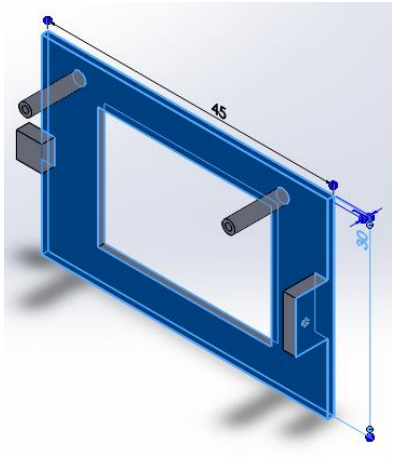


Fig. 72. Dimensiones de la tapa de la pulsera.
Realizado por el investigador.

Esta posee una abertura para la sujeción con tornillo, a la misma distancia que la abertura en la otra pieza.

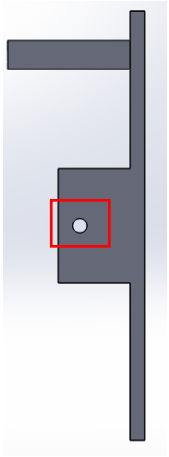


Fig. 73. Vista lateral izquierda del diseño de la tapa de la pulsera.
Realizado por el investigador.

Además, posee una ceja que encaja en la abertura de la otra pieza.

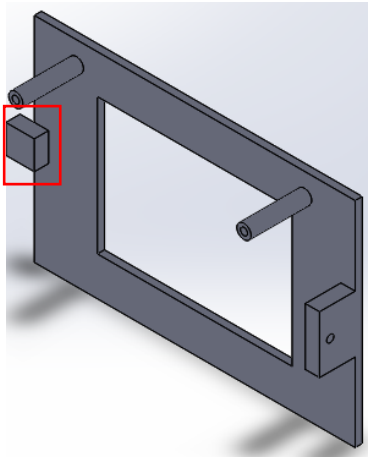


Fig. 74. Ceja de sujeción de la tapa de la pulsera.

Realizado por el investigador.

Y bases cilíndricas para la sujeción con la placa electrónica.

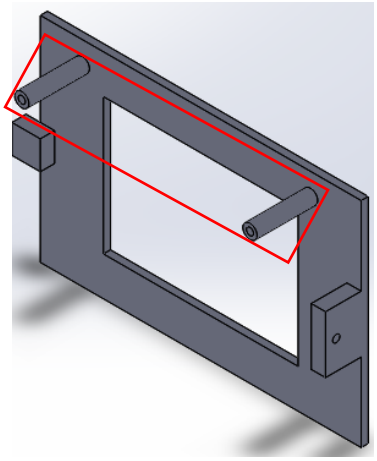


Fig. 75. Bases cilíndricas de la tapa para la sujeción de la placa electrónica.

Realizado por el investigador.

Al armar la pulsera y colocar el circuito en su interior, esta se verá como se muestra en la Fig. 76.

Fig. 76. Pulsera armada.

Realizado por el investigador.

Para evitar cualquier interferencia lumínica, se ha cocido tela de color negra alrededor del sensor MAX30100, como se ve en la Fig. 77.

Fig. 77. Protección de tela de color negra para el sensor MAX30100.

Realizado por el investigador.

3.1.1.7. Implementación final del proyecto

La implementación del proyecto se realizó en función a la estructura definida en la Fig. 13 de la sección 3.1.1.

Fig. 78. Implementación final del proyecto.

Realizado por el investigador.

Donde:

1. Es el router HG110 que brinda la conexión a internet de la empresa estatal CNT.
2. Es un televisor de marca LG con un puerto HDMI para la conexión del tv-box.
3. Tv-box con el sistema operativo Android TV 7.1.2.
4. Pulsera de medición de BPMs y SpO2 del usuario.
5. Control remoto del tv-box para la interacción con la aplicación.

3.1.1.8. Pruebas de funcionamiento

Para realizar las pruebas de funcionamiento se deben tomar en cuenta las direcciones de la tabla 14 de la sección 3.1.1.2. También se utilizará el "*log*" de Odoo al ejecutar el comando "*tail -f /var/log/odoo/odoo-server.log*" en el terminal de la maquina Ubuntu. Así se monitoreará cada petición HTTP mediante la librería de Python "*Werkzeug*", que se instaló junto con Odoo.

```
ubuntu@ip-172-31-23-168:~$ tail -f /var/log/odoo/odoo-server.log
```

Al abrir la aplicación se mostrará la pantalla de carga de la vista "*Splash*", y en el log de Odoo se visualiza la petición HTTP a la API "*/api/splash*" desde la dirección IP publica del usuario.

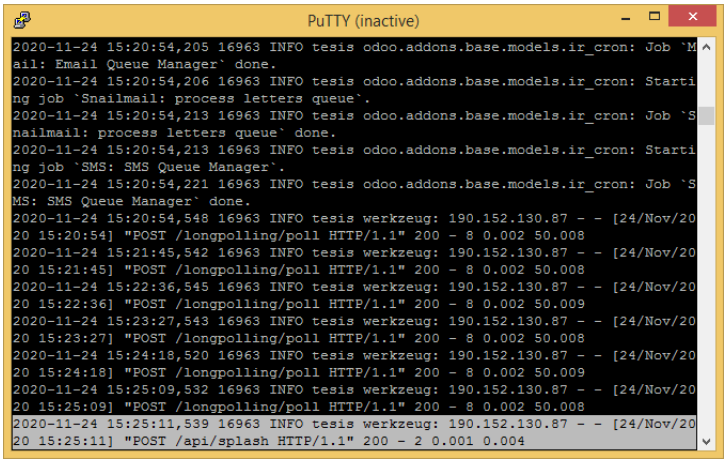


Fig. 79. Petición a la API "/api/splash" en el log de Odoo.

Realizado por el investigador.

Al renderizar la vista "*Usuarios*", la aplicación realizara una petición a la API "*/api/datos/idDispositivo*" desde la dirección IP publica del usuario, para consultar los usuarios que están ligados al dispositivo, como se ve en el log del Odoo. Esta petición se le realiza cada vez que el usuario navega dentro de ella, permitiendo actualizar el nombre del usuario en el caso de que haya sido editado en la vista "*Crear*". O en el caso de que un usuario haya sido eliminado.

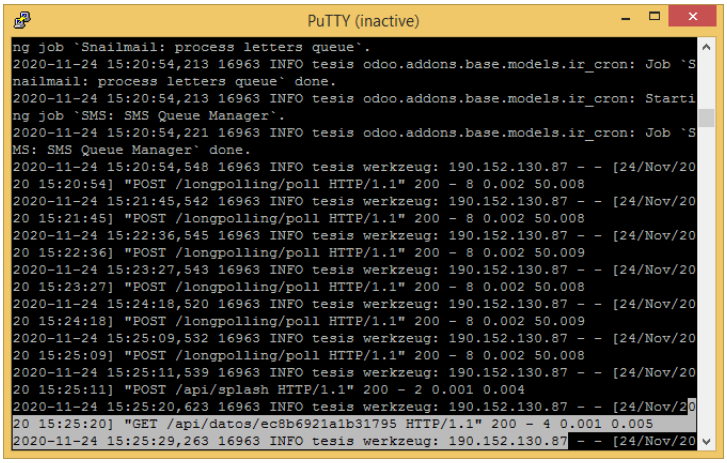


Fig. 80. Petición a la API "/api/datos/idDispositivo" en el log de Odoo.

Realizado por el investigador.

La vista renderizada para esta prueba es la que se muestra en la Fig. 81.

Fig. 81. Vista "*Usuarios*".

Realizado por el investigador.

Creación de un usuario

Para la creación de un nuevo usuario se debe presionar sobre el botón "*CREAR USUARIO*" y se mostrará la vista "*Crear*". Luego se llenan los datos del nuevo usuario.

Fig. 82. Vista "Crear" con los datos del nuevo usuario.

Realizado por el investigador.

Al presionar el botón "*CREAR*" se observa en el log que se hizo una petición HTTP a la "*/api/crear_usuario*" mediante el método POST desde la dirección IP publica del usuario.

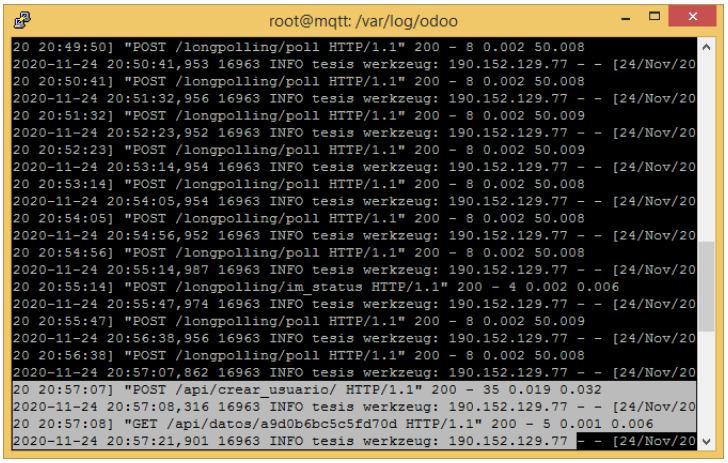


Fig. 83. Petición a la API "/api/crear_usuario" en el log de Odoo.

Realizado por el investigador.

Luego se realiza otra consulta a "*/api/datos/idDispositivo*" ya que se navega hacia la vista "*Usuarios*" de la aplicación, donde se observa el nuevo usuario agregado.

Fig. 84. Vista "Usuarios" con el nuevo usuario creado.

Realizado por el investigador.

Al seleccionar alguno de los usuarios, la aplicación realizará una petición a la API "*/api/prescripcion/idUsuario*" y a la API "*/api/paciente/idUsuario*" para consultar la última prescripción que se ha enviado y los datos del usuario respectivamente.

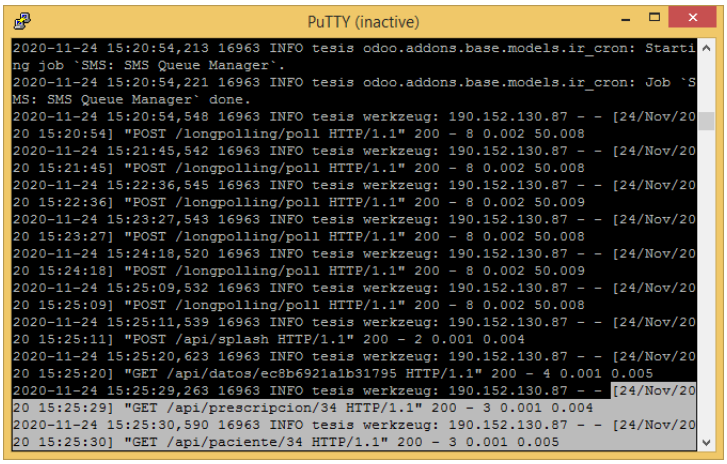


Fig. 85. Petición a la API "/api/prescripcion/idUsuario" y a la API "/api/paciente/idUsuario" en el log de Odoo.

Realizado por el investigador.

La vista renderizada se muestra en al Fig. 110. Como se ha creado un nuevo usuario, no hay una prescripción que se muestre.

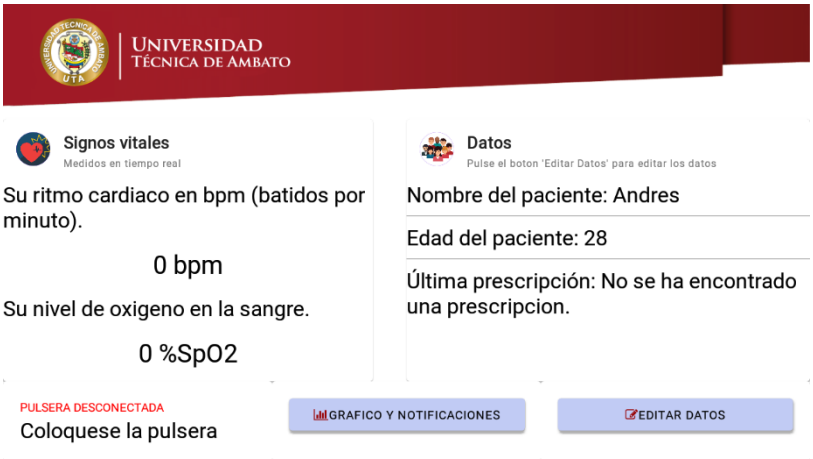


Fig. 86. Vista "*Datos*" renderizada del nuevo usuario creado.

Realizado por el investigador.

Modificar datos de un usuario

Si se selecciona el botón "*EDITAR DATOS*", este navegará a la vista "*Crear*" pero esta vez con los botones "*MODIFICAR*" y "*BORRAR USUARIO*".

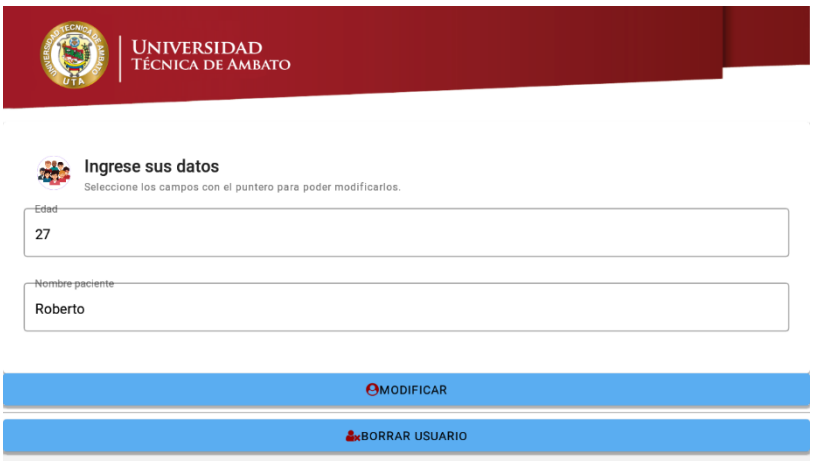


Fig. 87. Inserción de los datos modificados del usuario.

Realizado por el investigador.

Al modificar los datos y presionar le botón "*MODIFICAR*" la aplicación navegará a la vista "Datos", mostrando las modificaciones.

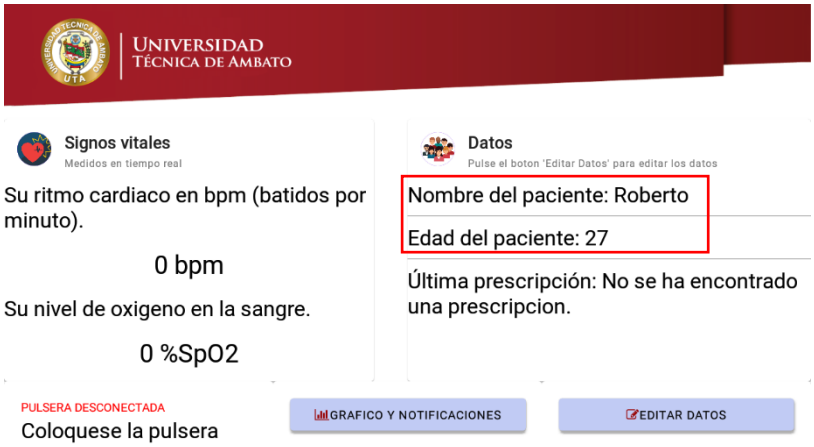


Fig. 88. Modificación de los datos del usuario.

Realizado por el investigador.

En el log se puede visualizar que se ha hecho una petición HTTP con el método PUT a "*/api/modificar_usuario*" desde la dirección IP del usuario. A continuación, se vuelven a realizar las peticiones a las APIs "*/api/prescripcion/idUsuario*" y "*/api/paciente/idUsuario*" para actualizar los datos del usuario.

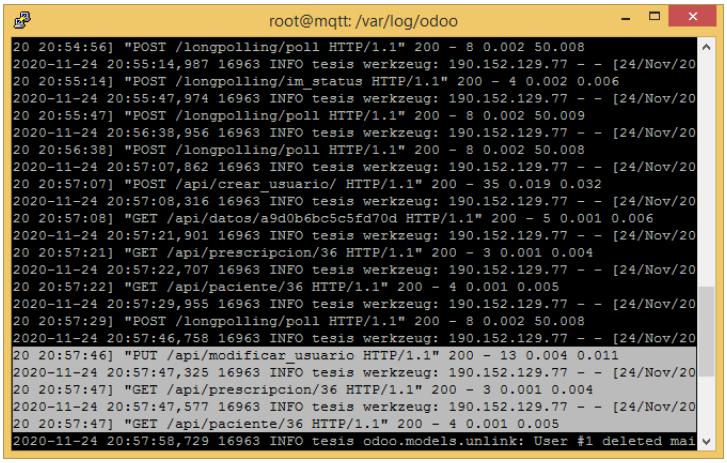


Fig. 89. Peticiones a las APIs de Odoo al modificar un usuario.

Realizado por el investigador.

Borrar un usuario

Para borrar el usuario, desde la vista "*Crear*" se presiona el botón "*BORRAR USUARIO*", entonces un mensaje de confirmación. Si se selecciona "*ACEPTAR*" la aplicación navegará a la vista "*Usuarios*", mostrando la lista de usuarios actualizada.

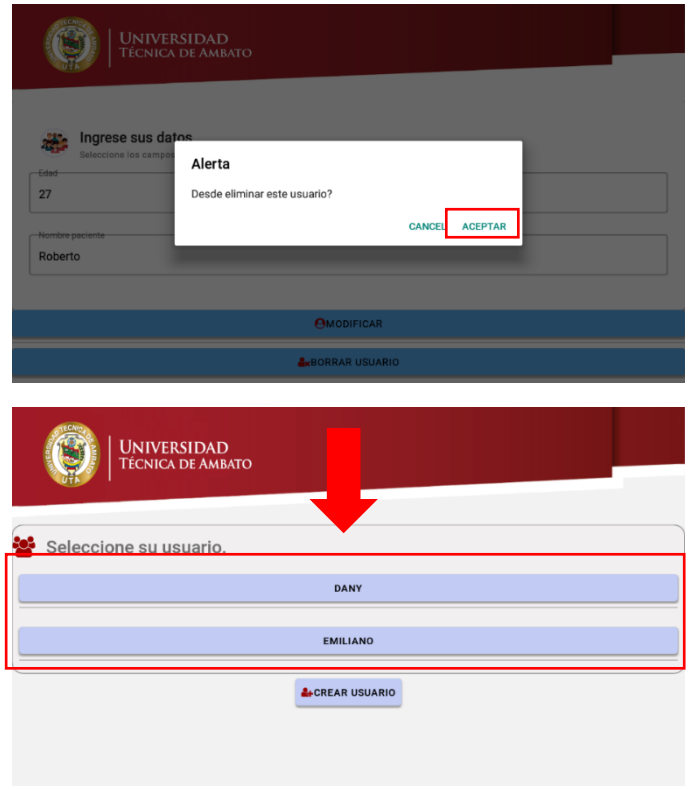


Fig. 90. Borrado de un usuario en la aplicación.

Realizado por el investigador.

En el "log" se observa que se realizó una petición HTTP a "*/api/borrar*" mediante el método POST desde la IP pública del usuario. Al final se realiza una petición a la API "*/api/datos/idDispositivo*" para actualizar la lista de los usuarios.

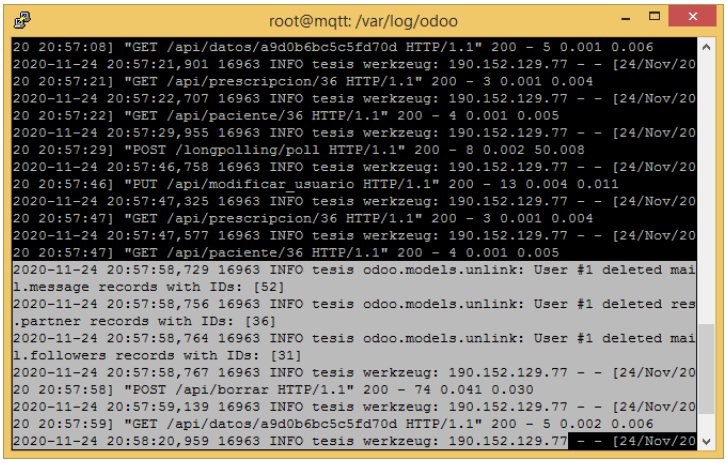


Fig. 91. Petición a la API "/api/borrar" en el log de Odoo.

Realizado por el investigador.

Gráfico y notificaciones

A presionar le botón "*GRAFICO Y NOTIFICACIONES*" de la vista "*Datos*" de cualquier usuario se realiza una consulta de los últimos datos registrados en la última

semana. En el caso de que no existan valores la aplicación mostrara un gráfico vacío y un mensaje “No hay datos para mostrar en las fechas ingresadas”.

Fig. 92. Gráfico vacío en la vista "Grafico" al no existir datos que mostrar.
Realizado por el investigador.

Al seleccionar un rango de fechas en el que existan datos y presionar el botón “*BUSCAR*”, la aplicación realizara una consulta HTTP con el método POST a la API “*/api/grafica*” y a su vez a la API “*/api/actNoti*” para actualizar el campo “*Mensaje leído*” en Odoo.

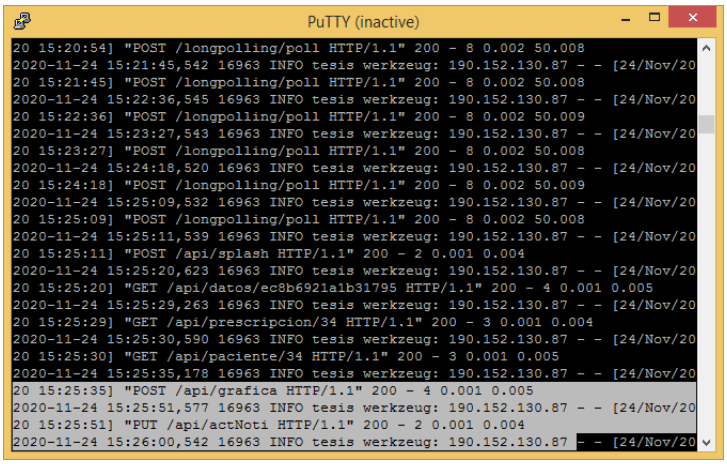


Fig. 93. Peticiones a las APIs “/api/grafica” y “/api/actNoti” en el log de Odoo.
Realizado por el investigador.

Renderizando todos los datos en la fecha seleccionada.

Fig. 94. Renderizado de los datos en el gráfico en la vista "Grafico".

Realizado por el investigador.

Medición de signos vitales

Para la medición de los signos vitales, el sensor debe estar sobre la muñeca debajo del dedo pulgar.

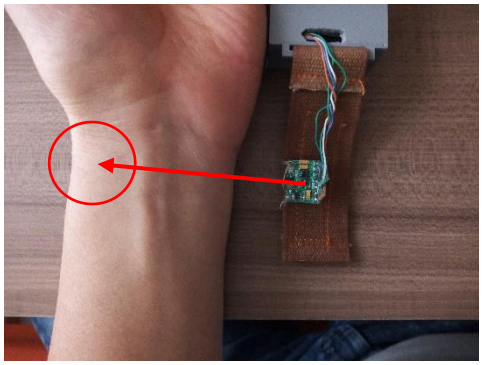


Fig. 95. Colocar la pulsera sobre la muñeca, debajo del dedo pulgar.

Realizado por el investigador.

Al encender la pulsera, esta mostrara la vista inicial la cual muestra un mensaje de "Esperando conectar". Se puede medir BPMs, SpO2 y nivel de batería en esta pantalla.

Fig. 96. Medición de datos en la pulsera sin conexión Bluetooth.

Realizado por el investigador.

La pulsera esperara hasta que el usuario seleccione el botón "*WEAREABLE-OXIMETER*" en la aplicación.

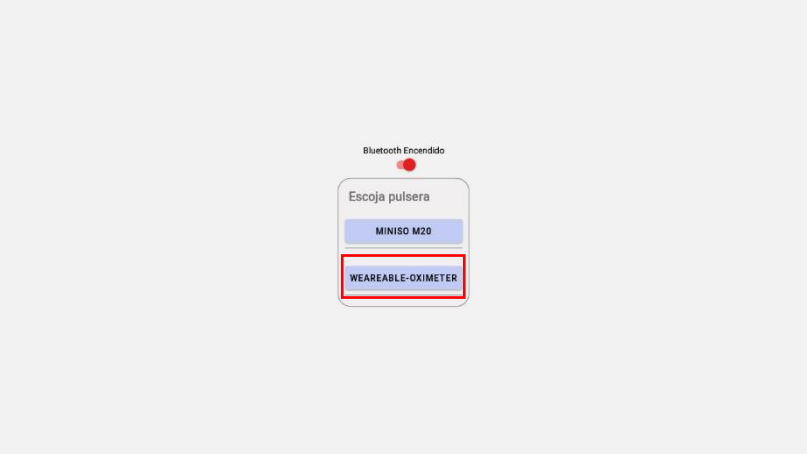


Fig. 97. Conexión Bluetooth entre la aplicación y la pulsera.

Realizado por el investigador.

Al seleccionar el botón, la aplicación intentara la conexión, mostrando la siguiente vista. Si se realiza la conexión, se mostrará un Toast con el mensaje "Conectado".

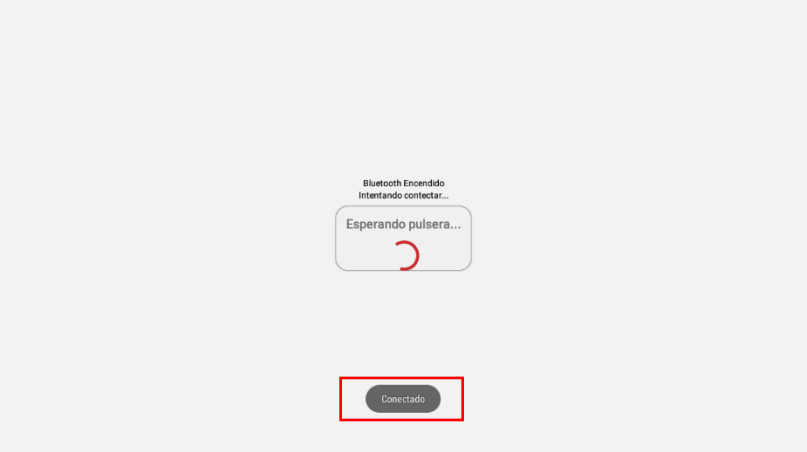


Fig. 98. Mensaje de verificación de la conexión bluetooth.

Realizado por el investigador.

Una vez conectado la pulsera, esta mostrará la pantalla de espera a que un usuario sea seleccionado en la aplicación. Aun si no se ha seleccionado un usuario, se podrán medir los signos vitales y el nivel de batería.

Fig. 99. Pantalla Oled una vez realiza la vinculación bluetooth.

Realizado por el investigador.

Una vez seleccionado un usuario dentro de la aplicación, la pulsera mostrara su nombre en la pantalla Oled y en la aplicación se comienza con la medición.

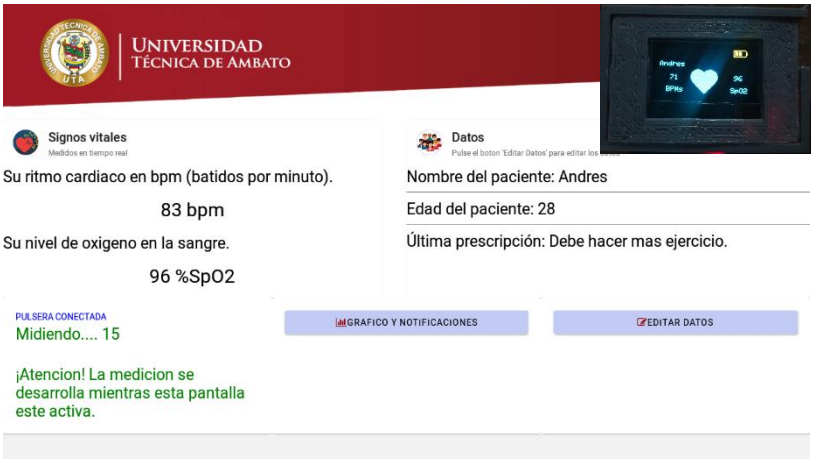


Fig. 100. Visualización de los datos en la pulsera y en la aplicación.

Realizado por el investigador.

Al terminar la medición, la aplicación mostrará un mensaje advirtiendo que se han subido con éxito los datos.

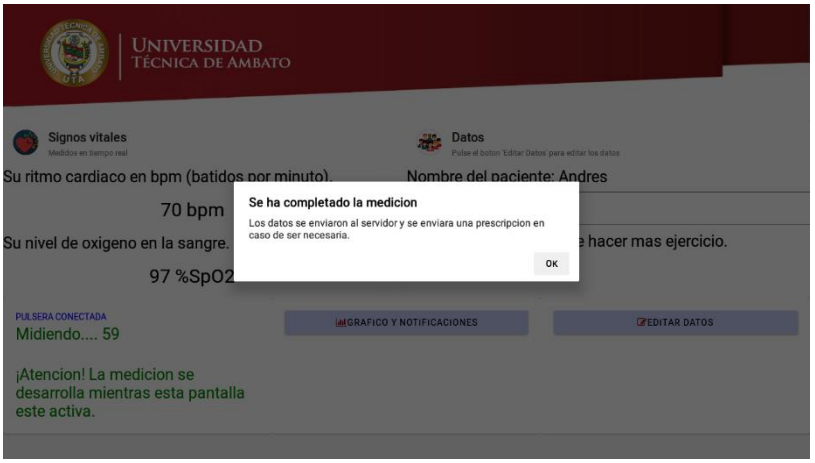


Fig. 101. Medición completa de los datos en la aplicación.

Realizado por el investigador.

En el log de Odoo se muestra la consulta a la API "*/api/datosPaciente*" desde la IP publica del usuario mediante el método POST.

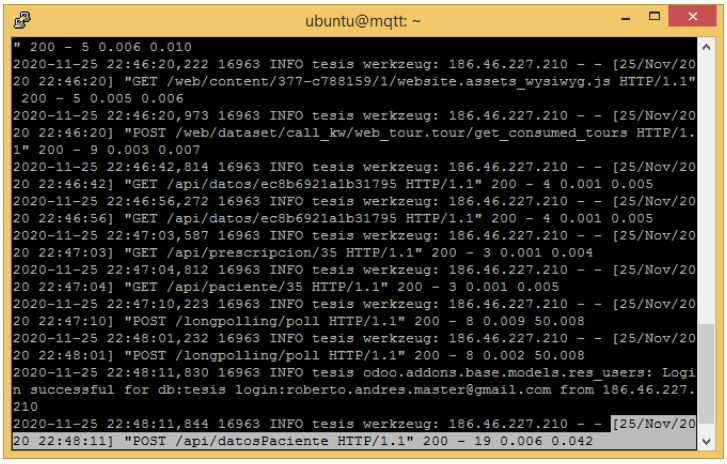


Fig. 102. Petición a la API "/api/datosPaciente" en el log de Odoo.

Realizado por el investigador.

Y en la tabla de signos vitales del usuario en la interfaz web de Odoo, se observan dichos datos.

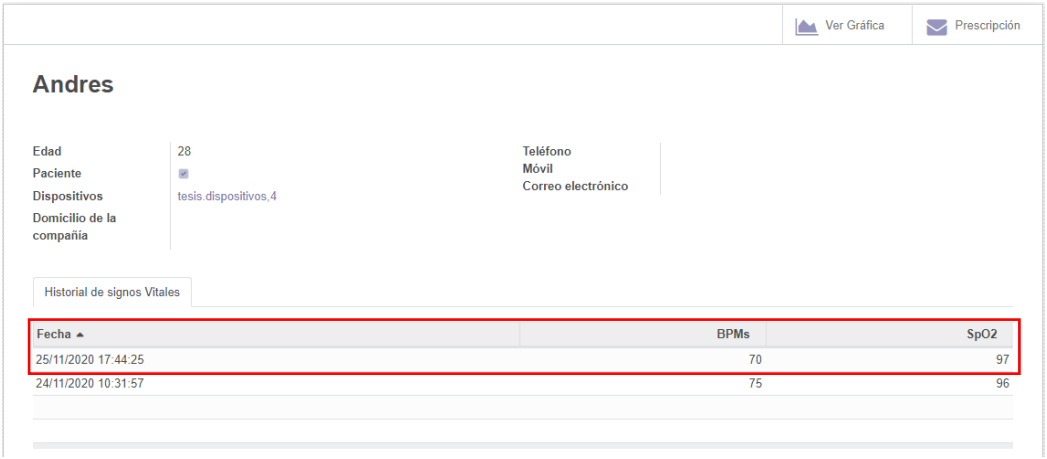


Fig. 103. Datos subidos a la tabla del usuario en Odoo.

Realizado por el investigador.

Notificaciones y prescripciones en Odoo

Para probar esta funcionalidad se simularán datos a ser enviados desde el software "Postman". Para activar una notificación se utilizarán el siguiente objeto:

```
{
    "res_paciente": 35,
    "presion": 76,
    "oxigeno": 94
}
```


En ambos casos activaran una notificación y mostraran un mensaje en la parte superior derecha de la interfaz web.

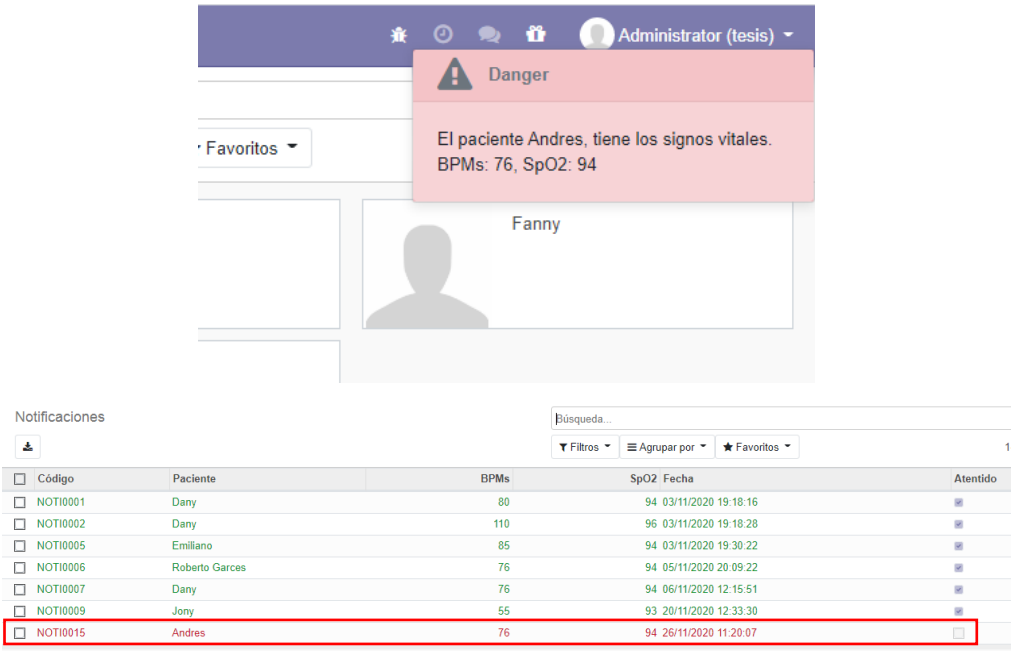


Fig. 104. Activación de las notificaciones en Odoo.
Realizado por el investigador.

Al dar clic sobre una notificación atendida, se mostrarán más detalles sobre la misma. En este caso no se muestra el botón atender ya que esta fue atendida, evitando errores en el registro de notificaciones y prescripciones atendidas y enviadas. Si se selecciona una notificación no atendida, se mostrará el botón "*Atender Notificación"* y el campo "*Atendido*" no estará marcado. Al presionar el botón "*Atender Notificación*" se mostrará el formulario de envío, donde se ingresa el texto a enviar al paciente. Además, se muestran los campos "*Paciente*" y "*Notificación*", los cuales se pueden modificar en caso de que se quiera enviar a otro paciente o atender otra notificación. Al guardar y enviar el mensaje, se mostrará un Toast en la aplicación, indicando que el nombre del paciente y el motivo de la notificación.

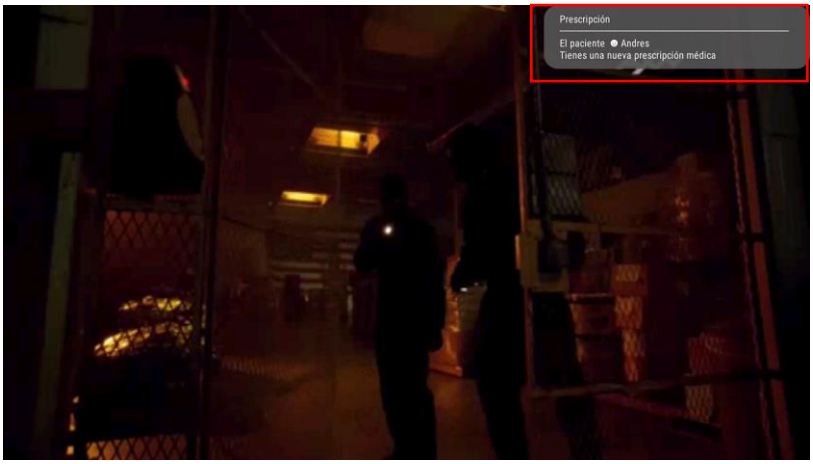


Fig. 105. Notificación de una nueva prescripción en el sistema Android TV. Realizado por el investigador.

Después la notificación pasará al estado de "Atendido". En la sección de prescripciones, se observa que la notificación que se acaba de enviar aún no ha sido leída por el usuario. Una vez que el usuario lea la notificación dentro de la vista de "*Grafico*" en la sección de notificaciones, esta aparecerá como leída.

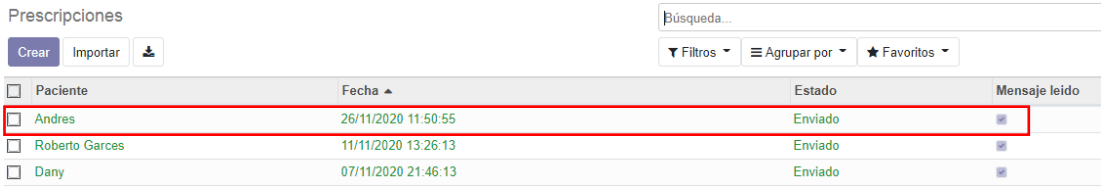


Fig. 106. Campo "Mensaje leído" de una prescripción leída por el usuario en Odoo. Realizado por el investigador.

3.1.1.9. Resultados

Latencias de la comunicación Bluetooth

Al realizar las mediciones por medio de bluetooth se capturaron los datos de latencia que se muestran en la Fig. 107, por medio de la herramienta de Android TV "*HCI Bluetooth snoop log*" en la sección de opciones de desarrollo. Estos datos representan paquetes recibidos con un tamaño de 21 bytes.

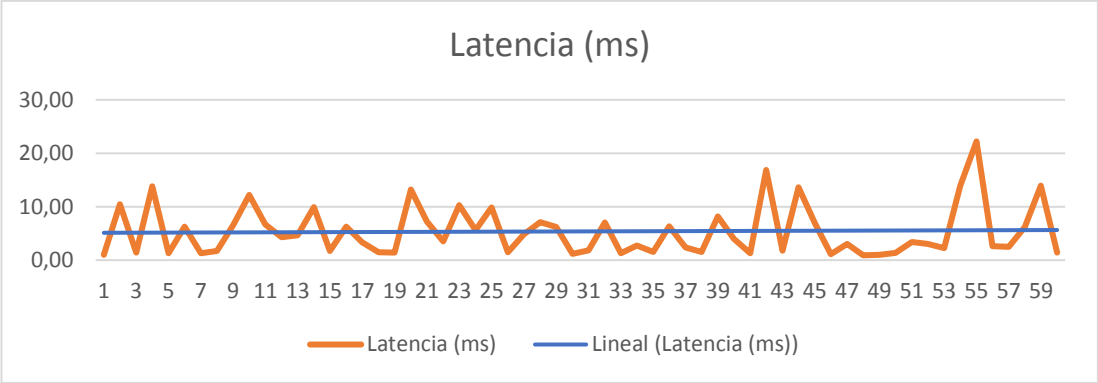


Fig. 107. Datos de latencia al medir los datos por Bluetooth.

Realizado por el investigador.

Al utilizar la herramienta de Excel "*Análisis de datos*", se obtuvo los datos de la tabla 21.

TABLA XXI. Resultados estadísticos de los datos de latencia de la transmisión Bluetooth.

Realizado por el investigador.

Latencia (ms)	
Media	5,37
Error típico	0,61
Mediana	3,456
Desviación estándar	4,75
Varianza de la muestra	22,56
Rango	21,32
Mínimo	0,90
Máximo	22,22
Suma	322,28
Cuenta (muestras)	60

El valor medio indica que la latencia se mantiene en 5.37 milisegundos durante el proceso de medición con una tendencia a aumentar con el tiempo. También se aprecia que la latencia puede tener valores de 0.90 milisegundos y 22.22 milisegundos durante

la medición, siendo casi imperceptibles para el usuario ya que son sumamente bajos, lo que se podría apreciar como la visualización en tiempo real de los datos en la pantalla del televisor.

Pruebas de Rendimiento de la aplicación de Odoo

Para estas pruebas se utilizó el software "*JMeter*". En esta sección se analizan los resultados obtenidos en dichas pruebas. Se han analizado las diferentes funcionalidades que la aplicación provee a los usuarios del grupo "*Monitores*" de la aplicación. Las pruebas se realizaron con las siguientes características de hardware que la capa gratuita de AWS ofrece:

- 1 CPU AMD i386, x86_64
- 1024 Mbytes de memoria RAM
- Ubuntu server 18.04

Pruebas de carga

Para estas pruebas se han definido un numero definido de usuarios, con un tiempo de consulta cada segundo y con una sola repetición entre cada consulta. Un resumen de los datos que se obtuvieron se puede apreciar en la tabla 22.

TABLA XXII. Resultados de las pruebas de carga a la aplicación Odoo. Realizado por el investigador.

Característica	10	100	1000
Numero de muestras	430	4300	43000
% error	0 %	1,61 %	84,84 %
Rendimiento	33,36/s	45,97/s	125,77/s
Media	525 ms	3206 ms	9166 ms

Prueba de estrés

A diferencia de las pruebas de carga, a la aplicación de Odoo se le aplica consultas de forma continua, sin interrupciones durante un periodo de tiempo, que en este caso fueron 30 minutos para cada prueba. Estas pruebas tienen las mismas configuraciones que las de carga, con la diferencia que las peticiones se hacen en un ciclo infinito. Al

finalizar las pruebas se obtuvo un resumen de los resultados mostrados en la tabla 23. Debido al alto consumo de recursos que provoca esta prueba, no se realiza la prueba de estrés con 1000 usuarios.

TABLA XXIII. Resultados de las pruebas de estrés a la aplicación Odoo. Realizado por el investigador.

Característica	10	100
Numero de muestras	44366	85642
% error	0,072 %	4,94 %
Rendimiento	30,82/s	44,12/s
Media	647 ms	4410 ms

A pesar del estrés que se le aplico al CPU, hubo una gran tasa de respuestas exitosas, lo que puede significar un punto de quiebre del hardware del servicio AWS con 100 usuarios. De estos resultados se puede concluir que el número idóneo de usuarios en el grupo "*Monitores*" puede estar en el rango de 1 a 100, con una eficiencia mínima del 95,06 %. Este número puede ser ampliado si se contrata un mejor hardware para el servicio, donde la cantidad de RAM es un factor importante. Al duplicar este valor, fácilmente se podría duplicar el número de usuarios, lo que significa que entre mejor es el hardware del servicio AWS, mayor es el rendimiento de la aplicación y menores tiempos de respuesta. Por otro lado, los núcleos del CPU pueden ayudar en la ejecución de varias tareas a la vez, por lo que su mejora se enfoca en si el servicio tiene varios procesos ejecutándose al mismo tiempo, por ello este trabajo se ha enfocado solo al funcionamiento de Odoo y sus componentes. En la Fig. 108 se muestra el porcentaje de uso de este, indicando que, en la carga de 100 usuarios su máximo rendimiento ya es utilizado, en cambio en la Fig. 109, con 10 usuarios el sistema tuvo una carga baja moderada.

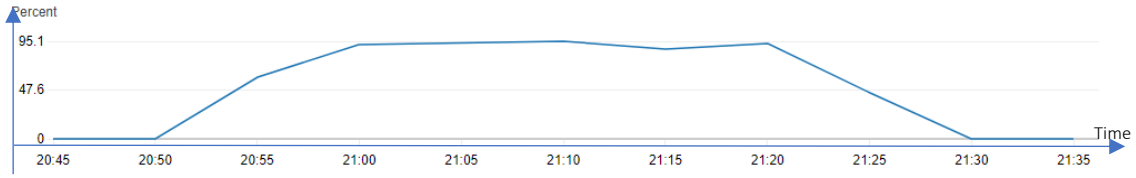


Fig. 108. Porcentaje de uso del CPU del servicio AWS en la prueba de estrés con 100 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

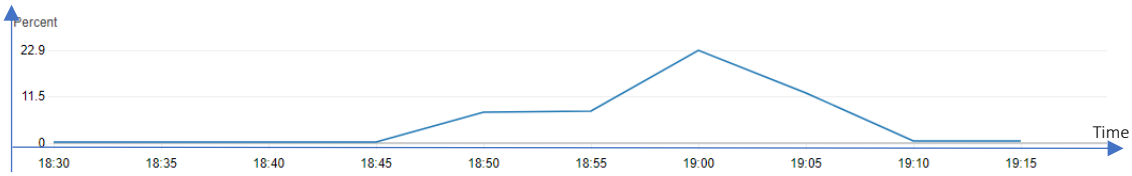


Fig. 109. Porcentaje de uso del CPU del servicio AWS en la prueba de estrés con 10 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

En estas pruebas además el consumo máximo de red fue de 3.29 Mbytes como se ve en la Fig. 110 y 838 Kbytes de consumo mínimo como se ve en la Fig. 111. Con este flujo de datos al día, no se cumple con el mínimo de tasa de transmisiones en el cual el servicio AWS tiene un cargo monetario, donde se cumple si se rebasa el terabyte de flujo de red por mes, es por ello por lo que el costo de servicio solo se enfoca en el pago de hardware.

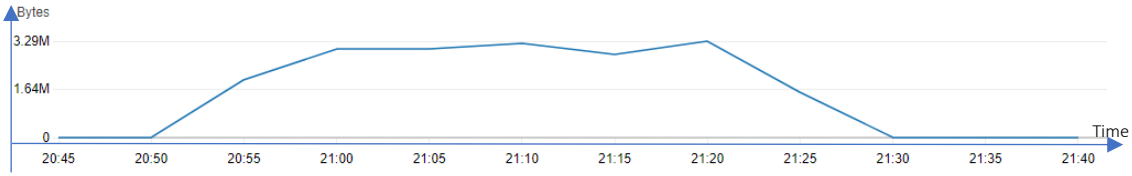


Fig. 110. Consumo de red durante la prueba de estrés con 100 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

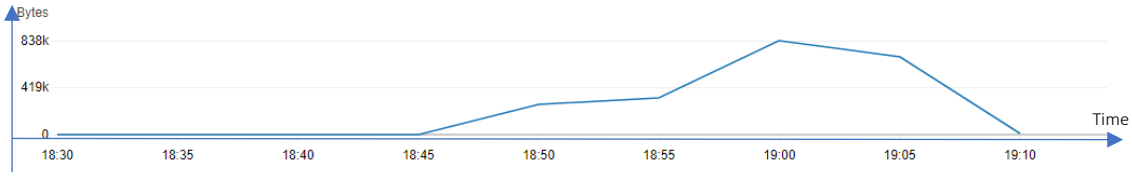


Fig. 111. Consumo de red durante la prueba de estrés con 10 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

Pruebas de carga APIs

Para estas pruebas se definieron las mismas configuraciones que para la aplicación. No se tomarán en cuenta todas las APIs ya que pueden causar errores al momento de lectura y provocar falsos negativos, como son en el caso de "*api/borrar*" y

"*api/modificar_usuario*". Al finalizar estas pruebas se resumieron con los datos mostrados en la tabla 24.

TABLA XXIV. Resultados de las pruebas de carga a las APIs.
Realizado por el investigador.

Característica	10	100	1000
Numero de muestras	100	1000	10000
% error	0 %	0,7 %	74,56 %
Rendimiento	21,11/s	11,27/s	47,71/s
Media	372 ms	2961 ms	5703 ms

Pruebas de estrés APIs

Al igual que para las pruebas anteriores, se toman en cuenta las mismas configuraciones a diferencia que las peticiones se realizan en un ciclo infinito durante alrededor de 30 minutos. Tampoco se tomaron en cuenta las APIs que se ignoraron en la prueba anterior. Los resultados se resumen en la tabla 25.

TABLA XXV. Resultados de las pruebas de estrés a las APIs.
Realizado por el investigador.

Característica	10	100
Numero de muestras	24599	57477
% error	0,081 %	4,27 %
Rendimiento	13,15/s	27,41/s
Media	918 ms	3360 ms

Durante las pruebas, el servicio de AWS tuvo un consumo de CPU maximo del 91.8% como se ve en la Fig. 112. Por otro lado, el consumo con 10 usuarios fue del 60,2% como se ve en la Fig. 113.

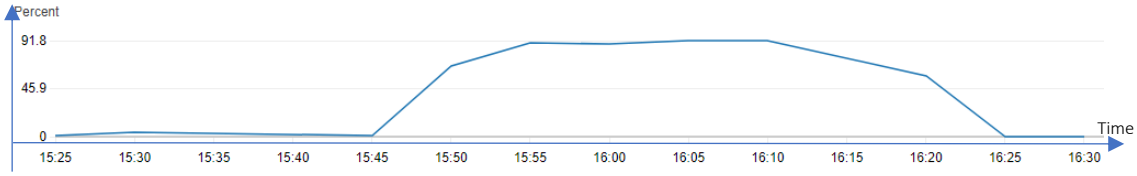


Fig. 112. Porcentaje de uso máximo del CPU del servicio AWS en las pruebas de estrés con 100 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

Fig. 113. Porcentaje de uso máximo del CPU del servicio AWS en las pruebas de estrés con 10 usuarios.

Fuente: "CloudWatch Management Console" de AWS.

A pesar del estrés que se le aplico a la aplicación hubo una alta tasa de respuestas exitosas y buenos niveles de rendimineto, pero las limitaciones de hardware aumentan los tiempos de espera y limita el número de peticiones que se pueden realizar. Para la utilización de las APIs se puede dimensionar hasta 100 usuarios, con una efectividad mínima del 95,73 %, o hasta 130 usuarios con una efectividad mínima del 93,16 %, esto en función a otras pruebas de estrés realizadas con esta cantidad de usuarios. Por demas, los datos subidos durante las pruebas se escribieron en la base de datos sin errores, es decir que los datos subidos llegaron al servicio AWS con 100% de efectividad. Al igual que en las pruebas de la aplicación de Odoo, aquí se evidencia un punto de quiebre del hardware con 100 usuarios, en donde si las caracteristicas de hardware mejoran, los tiempos de respuesta y número de usuarios lograría ampliarse, teniendo como referencia los datos obtenidos en estas pruebas. En este caso, el aumento de número de núcleos podría significar en un mejor manejo de la aplicación y de las APIs ejecutandose a la vez, pero para esta aplicación el numero ideal de usarios sería de 10 Monitores para la aplicación Odoo, y 100 usuarios para las APIs, que son los usuarios que utilizan la aplicaion en el sistema Android TV, con un tiempo medio de espera mínimo de 918 milisegundos y un máximo de 3360 milisegundos.

El consumo de red durante estas pruebas fue de un máximo de 1,46 Mbytes como se ve en la Fig. 114. Para un total de 10 usuarios el consumo de 818 Kbytes como se ve en la Fig. 115. Estos valores combinados con la tasa de transferencia de las pruebas de carga suman los 47,5 Mbytes por día, significando una cantidad total de 142,5 Mbytes

por mes, lo cual no represento un costo monetario en el servicio AWS, ya que no se pasa del límite mínimo de 1Tbytes por mes.

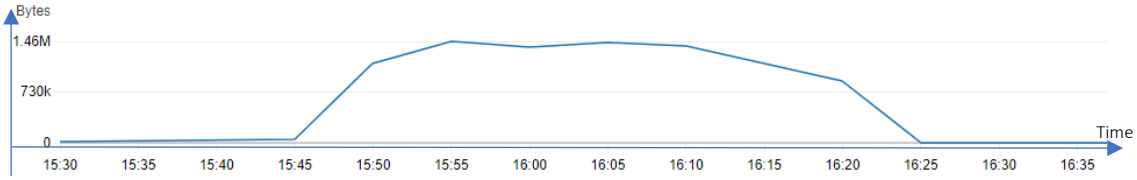


Fig. 114. Consumo de red máximo durante las pruebas de carga de las APIs con 100 usuarios. Fuente: "CloudWatch Management Console" de AWS.

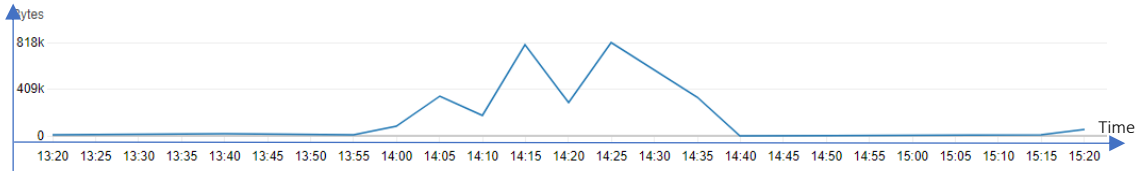
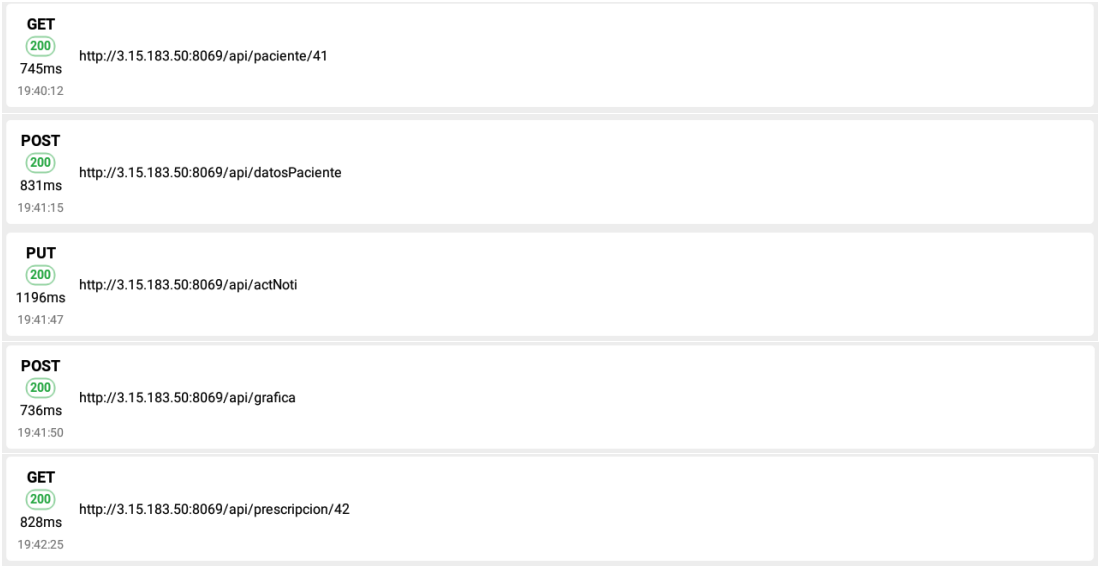


Fig. 115. Consumo de red máximo durante las pruebas de carga de las APIs con 10 usuarios. Fuente: "CloudWatch Management Console" de AWS.

Latencias de respuesta de la aplicación para Android TV

Para determinar los tiempos de respuesta se ha instalado la librería "*react-native-network-logger*" que permite monitorear las peticiones que la aplicación realiza y sus tiempos de respuesta. Estos datos fueron medidos con un estrés de 50 usuarios a las APIs, intentando simular un ambiente real en la utilización de la aplicación. Los resultados de la librería se muestran en la Fig. 116.



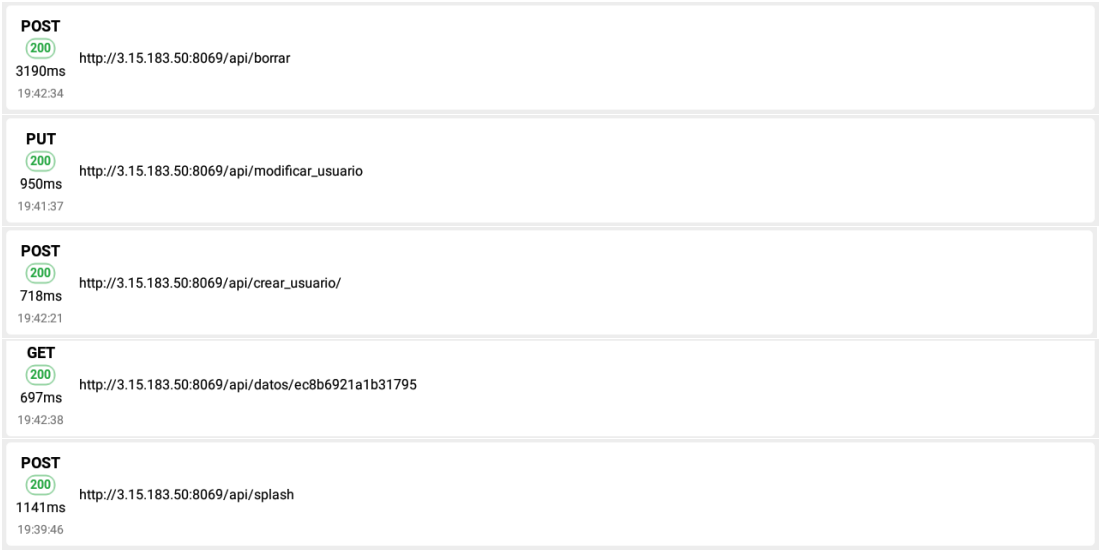


Fig. 116. Tiempos de respuesta adquiridos por la librería " *react-native-network-logger* " dentro de la aplicación para Android TV.
Realizado por el investigador.

El tiempo de llegada del WebSocket se obtuvo al correr el logger de Odoo en modo "*debug*", en donde se aprecia el tiempo en el que se envió el mensaje desde Odoo, como en la Fig. 117.

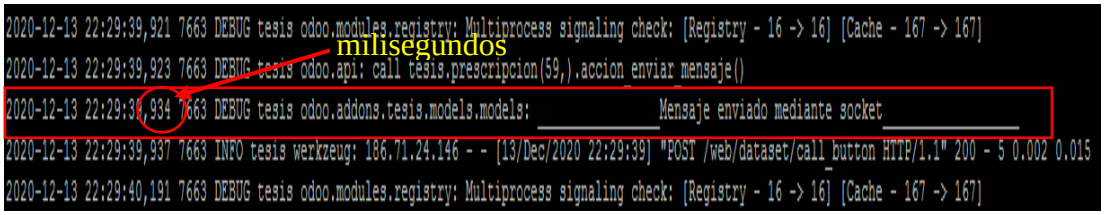


Fig. 117. Tiempo de salida del mensaje por socket en el log de Odoo.

Realizado por el investigador.

Y el tiempo de llegada se capturo con el software Wireshark, como se ve en la Fig. 118.

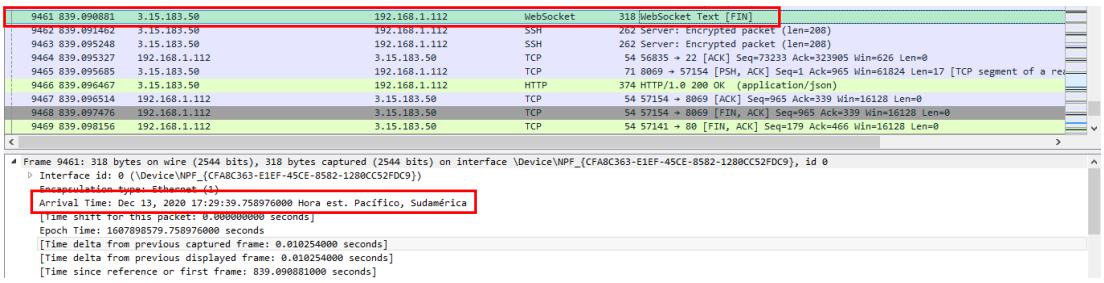


Fig. 118. Tiempo de llegada del mensaje por WebSocket.

Realizado por el investigador.

En esta medición la IP del servidor es "*3.15.183.50:8069*" ya que antes de realizar las pruebas la instancia se reinició por motivos de consumo mínimo de horas por el servicio de AWS en la capa gratuita. En el log de Odoo, la hora está configurada de

acuerdo con la ubicación del servidor la cual es "*Etc/UTC (UTC, +0000)*", entonces, al hacer la diferencia de minutos y segundos el resultado es 175 milisegundos. La curva de los tiempos de respuesta de la aplicación se muestra en la Fig. 119.

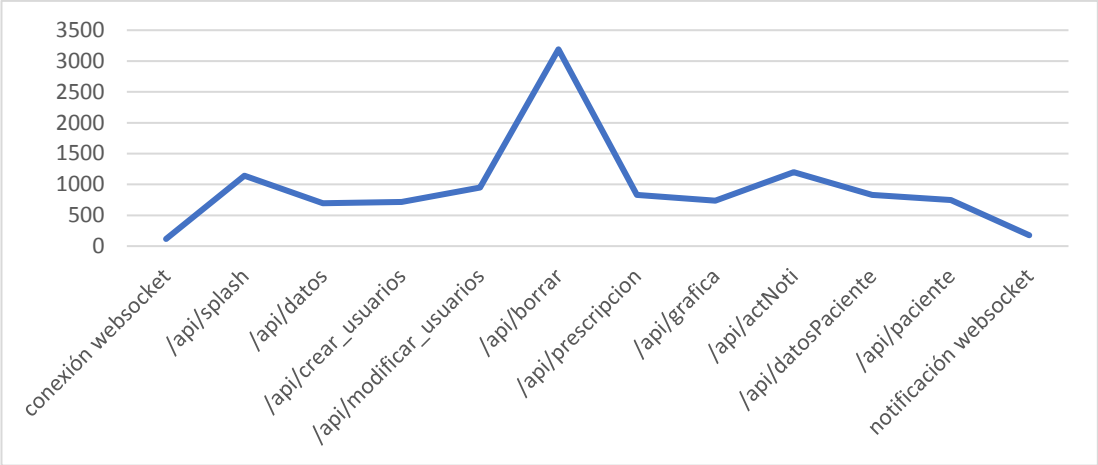


Fig. 119. Curva de los tiempos de respuesta de la aplicación de Android TV. Realizado por el investigador.

El análisis estadístico de estos tiempos se puede ver en la tabla 26.

TABLA XXVI. Análisis estadístico de los tiempos de respuesta de la aplicación de Android TV. Realizado por el investigador.

Resultados de latencia (ms)	
Media	943,58
Error típico	224,37
Mediana	786,5
Desviación estándar	777,23
Mínimo	116
Máximo	3190

Un Boxplot con los datos obtenidos en esta prueba se puede ver en la Fig. 120.

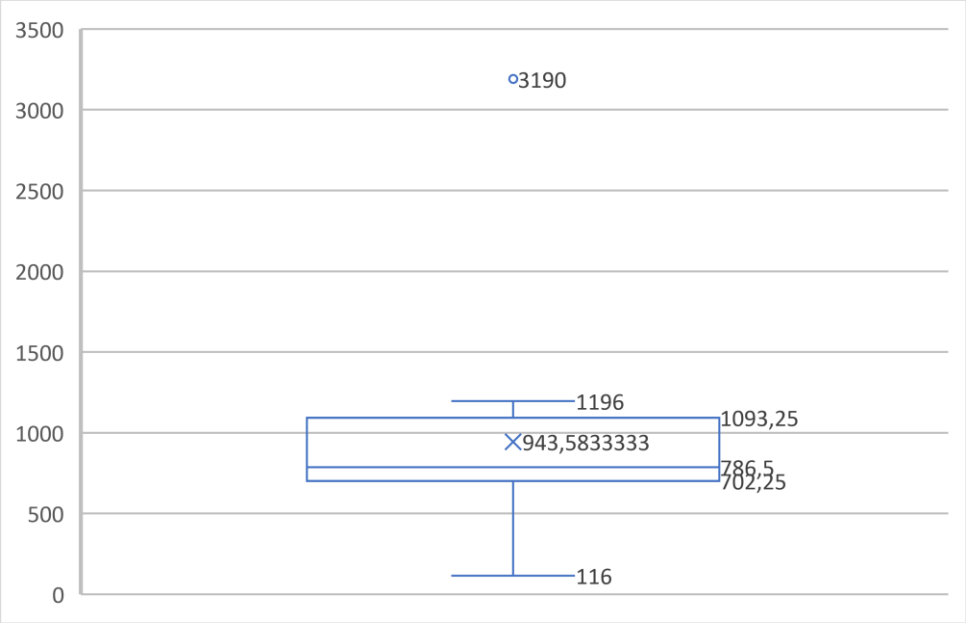


Fig. 120. Boxplot de las latencias de respuesta de la aplicación para el sistema Android TV. Realizado por el investigador.

Ya que Odoo envía una respuesta HTTP "*200*" a la aplicación una vez que el proceso de consulta, actualización, borrado o subida de datos fue exitosamente realizado en la base de datos, estos serían los tiempos máximos de latencia para la navegación entre las vistas de la aplicación. Según la tabla 26, la media de estas respuestas es de 943,58 milisegundos, con una variancia dentro del rango de 777,23 milisegundos. El tiempo de respuesta más corto resulto ser la conexión para la recepción de notificaciones por WebSocket, siendo de 116 milisegundos, y la más alta el borrado de usuarios, con un tiempo de 3190 milisegundos. Según el Boxplot de la Fig. 120, el retardo con menor porcentaje de ocurrencia es el de 3190 milisegundos, y los que tiene un 75% o más de ocurrencia son aquellos datos que se encuentran en el rango de 702,5 y 1196 milisegundos, con una media de 943,58 milisegundos.

Análisis de las encuestas

Para este análisis se utilizó la encuesta que se muestra en el ANEXO U. La muestra fueron 40 profesionales de la salud a los que se les presento el proyecto desarrollado y después respondieron a la encuesta. Los datos son analizados mediante el método conocido como "*Boxplot*". Los resultados se pueden ver en la Fig. 121.

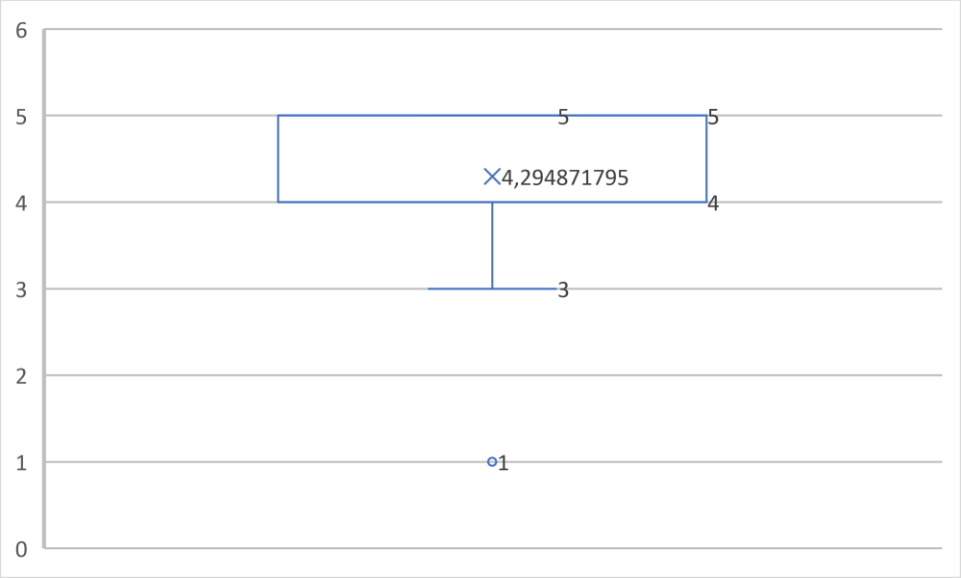


Fig. 121. Boxplot de las encuestas realizadas.

Realizado por el investigador.

El objetivo de la primera pregunta es el de saber que tan importantes son los datos mostrados por el sistema, en el cual 92,3% dieron una calificación entre 4 y 5, donde el 73,2% son calificaciones de 5. Actualmente el nivel de SpO2 puede determinar si una persona sufre del virus COVID-19, siendo un dato esencial para la actual pandemia. El objetivo de la segunda pregunta es el de saber que tan útiles son las herramientas presentadas por el sistema, en el cual 76,9% de las personas dieron una calificación entre 4 y 5, donde el 53,9% son calificaciones de 5. Estas herramientas evitan que los pacientes y médicos se movilicen en casos de riesgo epidemiológico, como la actual pandemia del COVID-19, o en casos donde el paciente no puede movilizarse al centro de salud más cercano, debido a distintas afecciones o por su ubicación geográfica.

El objetivo de la tercera pregunta es el de saber si el manejo de la plataforma es sencillo y de fácil manejo para el usuario, en el cual 80,8% de las personas dieron una calificación entre 4 y 5, donde el 34,6% son calificaciones de 5. En el gráfico se observa un valor atípico mínimo de 1, el cual tiene un valor del 3,8% en cada pregunta, y debido a este bajo porcentaje no se toma en cuenta en ningún cuartil.

El objetivo de la última pregunta es el saber si los profesionales de la salud implementarían el sistema propuesto en sus lugares de trabajo. Los resultados se pueden apreciar en el diagrama de pastel de la Fig. 122.

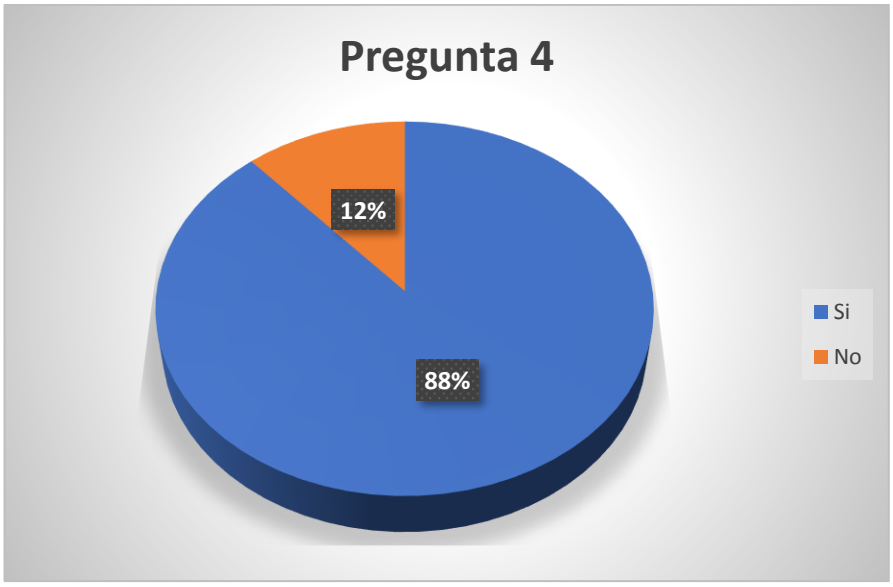


Fig. 122. Diagrama de pastel con los resultados de la pregunta 4.

Realizado por el investigador.

Del total de la muestra, una gran mayoría implementaría este sistema en sus lugares de trabajo, siendo un total del 88%. Estos datos indican una buena aceptación del sistema dentro del sector de salud, donde más del 75% de encuestados dieron calificaciones entre 4 y 5.

3.1.10. Costos del prototipo

Precios de hardware

Para este proyecto se adquirió un tv-box de la marca "*Speed Box Fusion*" el cual tiene soporte bluetooth y wifi para el correcto funcionamiento de la aplicación. En la capa se sensado se utilizaron diferentes materiales y procesos los cuales se definen en la tabla 28.

TABLA XXVII. Detalles de precios de hardware para la realización del proyecto.

Realizado por el investigador.

Recurso	Precio unitario	Cantidad	Total
Arduino Mini Pro 5V 16MHz	$ 8,00	1	$ 8,00
Batería 3,7 V - 450 mA	$ 15,00	1	$ 15,00
Circuito de carga de baterías	$ 1,60	1	$ 1,60
Resistencias SMD	$ 0,10	2	$ 0,20

Pantalla OLED 128x64	$	8,00	1	$	8,00
Sensor MAX30100	$	8,00	1	$	8,00
Interruptor de encendido	$	0,50	1	$	0,50
Hora de impresión 3D	$	1,50	2	$	3,00
Baquelita doble cara	$	3,50	1	$	3,50
Estaño (metros)	$	1,00	1	$	1,00
Speed Box Fusion Android TV	$	149,00	1	$	149,00
Total				$	197,80

Precios de Software

Se utilizó un servicio AWS bajo demanda Linux, el cual maneja una instancia EC2. Según la consola de AWS, el precio para este servicio es de 0.0116 USD por hora de uso. Esto significaría un precio de operación mensual de 8.468 USD. Otros servicios del mismo tipo, pero con distintas características se pueden revisar en el ANEXO V. Para más información sobre el precio de las instancias en los distintos servicios de AWS se puede consultar la página oficial. Odoo por otro lado, es un software libre y sin precio de adquisición, por lo que no se toma en cuanta en el precio final de esta sección. Otro factor que tomar en cuenta son los precios del servicio de AWS en la trasferencia de datos. En este caso, se midió el tamaño del paquete de cada petición gracias al software Postman, como se indica en la Fig. 123.

Fig. 123. Medición del tamaño del paquete en cada petición.

Realizado por el investigador.

Entonces, cada vez que un usuario realiza una petición consume 420 Bytes. Según el INEC, en el ecuador existen 3.9 personas por familia y suponiendo que existen 1000 dispositivos registrados, equivaldría a un total de 3900 usuarios. Además, suponiendo en el peor de los casos que un usuario realiza 3 mediciones de sus signos diarios, que recibe 3 notificaciones, y que el número de APIs es 10, cada usuario realizaría 60 peticiones cada día, 30 en mediciones y 30 en consulta de notificaciones. Esto equivale a un consumo de 25.2 Kbytes por usuario, lo que equivale a 0.09828 Gbytes en total. Según la página oficial de AWS [44], esto no equivaldría un costo alguno, ya que no se supera la transferencia de 1 Gbyte por mes.

Precio mano de obra

Según la página web del ministerio de trabajo, un ingeniero en electrónica podría aspirar a un sueldo de 430 USD mensuales [45], y suponiendo que se invirtieron 6 meses de implementación del proyecto, equivaldría a un total de 2580 USD en total.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

Al finalizar el presente trabajo de titulación se han obtenido las siguientes conclusiones.

- Una vez analizadas las tecnologías disponibles en lo que concierna a la realización de la aplicación para el ambiente Smart TV, React Native resultó ser el más viable debido a que su desarrollo está pensado en la multiplataforma. Con pocos cambios de código, y en algunos casos ninguno, la aplicación desarrollada puede ser implementada en sistemas móviles iOS y Android, sistemas de sobremesa XBOX, Android TV y Apple TV, llegando a brindar todas las funcionalidades de la telemedicina del sistema implementado a muchas más personas.

- Durante las pruebas de rendimiento realizadas tanto a la aplicación de Odoo como a las APIs, el punto crítico llega con los 100 usuarios. En estas condiciones el porcentaje de respuestas exitosas alcanzan un mínimo del 95,73% con un promedio de 3360 milisegundos de latencia, sin sobrepasar la tasa mínima de red que el servicio de AWS proporciona en su capa gratuita.

- En la interfaz realizada, para simular un ambiente real se realizaron pruebas de estrés a las APIs con 50 usuarios mientras se capturaban los tiempos de respuesta de la aplicación en el sistema Android TV. El tiempo medio de respuesta es de 943,58 milisegundos tanto en peticiones a las APIs como con la comunicación mediante WebSocket. Por otro lado, las latencias en las transmisiones por bluetooth tienen un promedio de 5,37 milisegundos, y un máximo de 22,22 milisegundos durante los 60 segundos que dura la medición de BPMs y SpO2.

- Se realizó una encuesta a 40 profesionales de la salud que valoraron la aplicación desarrollada. El 92,4% piensan que los datos mostrados por el sistema son útiles para realizar una valoración médica del usuario; el 76,9% creen que las

herramientas son útiles para brindar un servicio de telemedicina; el 80,8% creen que la aplicación es sencilla en su manejo y el 88,5% estarían dispuestos a implementar este sistema en su lugar de trabajo.

- El trabajo de investigación realizado se enfocó en la medición de BPMs y SpO2, ya que son datos comúnmente utilizados en consultas médicas y pueden aumentar significativamente su efectividad. Con el fin de monitorear los signos vitales del paciente se ha diseñado una pulsera Weareable que contiene un circuito electrónico capaz de sensar los BPMs y niveles de SpO2 en la sangre del paciente, los cuales son enviados a través de bluetooth a la aplicación de Android TV, para luego ser subidos a la base de datos en Odoo por medio del protocolo TCP/IP desde la aplicación de Android TV, donde son procesados y consultados por los monitores. De esta forma se reducen gastos operativos médicos, ya que el paciente puede recibir prescripciones a través de la interfaz de la aplicación para Android TV.

4.2. Recomendaciones

En base a la implementación de este trabajo de titulación se ha encontrado diferentes recomendaciones en este proceso.

- Para la implementación de las APIs en Odoo, se las puede realizar en una base de datos local para una ágil identificación y corrección de errores. Después se puede exportar la base de datos creada de forma local e implementarla una vez instalada la instancia de Amazon.

- Se puede utilizar la herramienta de simulación de Android Studio para el diseño de la interfaz sin necesidad de generar el archivo APK cada vez que se realizan cambios. Esta herramienta es útil para probar el código y comprobar su funcionamiento. Se debe tomar en cuenta que la virtualización debe estar activa en la BIOS del computador.

- Para la instalación del software de administración de la nube Odoo y sus librerías es necesaria una velocidad de conexión a internet de un ancho de banda mínima

de 10 Mbps, ya que pueden existir problemas en la instalación, generando problemas al ejecutar el servicio de Odoo.

- La impresión en 3D de la pulsera Weareable se recomienda imprimirla con un ancho de capa de 0.12 mm para darle mayor definición y rigidez estructural.

- Es recomendable implementar el circuito electrónico de la pulsera Weareable en dos capas para minimizar su tamaño.

BIBLIOGRAFÍA

[1] Universidad Simón Bolívar *et al.*, «Internet of Things and Home-Centered Health», *Salud Uninorte*, vol. 32, n.º 2, pp. 337-351, may 2016, doi: 10.14482/sun.32.2.8954.

[2] E. Y. P. Castaño, L. C. Carvajal, J. J. B. García, y Y. S. P. Rengifo, «Estado actual de la telemedicina: una revisión de literatura* Current state of telemedicine: a literature review», n.º 20, p. 16.

[3] «Saludsa», *Saludsa*. https://www.saludsa.com/consulta_medica_en_linea (accedido dic. 17, 2020).

[4] «Gracias a la telemedicina, consultas no se detienen durante emergencia», *El Comercio*. http://www.elcomercio.com/actualidad/telemedicina-consultas-emergencia-salud-coronavirus.html (accedido dic. 17, 2020).

[5] U. Albalawi y S. Joshi, «Secure and trusted telemedicine in Internet of Things IoT», en *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Singapore, feb. 2018, pp. 30-34, doi: 10.1109/WF-IoT.2018.8355206.

[6] C. R. Costa, L. E. Anido-Rifon, y M. J. Fernandez-Iglesias, «An Open Architecture to Support Social and Health Services in a Smart TV Environment», *IEEE J. Biomed. Health Inform.*, vol. 21, n.º 2, pp. 549-560, mar. 2017, doi: 10.1109/JBHI.2016.2525725.

[7] G. Pires *et al.*, «VITASENIOR-MT: a telehealth solution for the elderly focused on the interaction with TV», en *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Ostrava, sep. 2018, pp. 1-6, doi: 10.1109/HealthCom.2018.8531126.

[8] N. M. Abdelsamee y A. Algarni, «IOT based solution for Teleconsulting Cardiac Patients In Saudi Arabia», en *2018 21st Saudi Computer Society National Computer Conference (NCC)*, Riyadh, abr. 2018, pp. 1-4, doi: 10.1109/NCG.2018.8593156.

[9] H. Bhojwani, G. K. Sain, y G. P. Sharma, «A Hybrid Connectivity Oriented Telemedician System For Indian Landscape Using Raspberry Pi SBC & IOT», en *2018 3rd Technology Innovation Management and Engineering Science*

International Conference (TIMES-iCON), Bangkok, Thailand, dic. 2018, pp. 1-5, doi: 10.1109/TIMES-iCON.2018.8621799.

[10] J.-C. Liau y C.-Y. Ho, «Intelligence IoT(Internal of Things) Telemedicine Health Care Space System for the Elderly Living Alone», en *2019 IEEE Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)*, Okinawa, Japan, may 2019, pp. 13-14, doi: 10.1109/ECBIOS.2019.8807821.

[11] A. S. Márquez y M. C. C. Vázquez, «Desarrollo de aplicaciones para SMART TV usando la tecnologia Android TV», p. 99.

[12] J.-W. Jeong, H.-K. Hong, y D.-H. Lee, «Ontology-based automatic video annotation technique in smart TV environment», *IEEE Trans. Consum. Electron.*, vol. 57, n.º 4, pp. 1830-1836, nov. 2011, doi: 10.1109/TCE.2011.6131160.

[13] W.-P. Lee, C. Kaoli, y J.-Y. Huang, «A smart TV system with body-gesture control, tag-based rating and context-aware recommendation», *Knowl.-Based Syst.*, vol. 56, pp. 167-178, ene. 2014, doi: 10.1016/j.knosys.2013.11.007.

[14] I. Anghelache, G. Kostopoulos, S. López, Y. Martínez, C. Rivas, y I. Žilavec, «Providing ICT-BASED formal and infomal care at home», vol. aal-2014-171, pp. 1-162, mar. 2019.

[15] M. Yener, «Expert Android® Studio», p. 456.

[16] «tvOS», *Apple Developer*. https://developer.apple.com/tvos/ (accedido dic. 17, 2020).

[17] «About | Tizen». https://www.tizen.org/about (accedido dic. 17, 2020).

[18] «Fire OS Overview | Amazon Fire TV». https://developer.amazon.com/docs/fire-tv/fire-os-overview.html (accedido dic. 17, 2020).

[19] «Global smart TV operating system share 2018», *Statista*. https://www.statista.com/statistics/882062/worldwide-smart-tv-operating-system-share/ (accedido dic. 17, 2020).

[20] W. Danielsson, «React Native application development», p. 70.

[21] N. N. Chawande, «A Review on Android Operating System with its Security Features», vol. 03, n.º 07, p. 5.

[22] B. Michéle, «Digital Television and Smart TVs», en *Smart TV Security*, Cham: Springer International Publishing, 2015, pp. 1-13.

[23] «La TV ensamblada en el país es más competitiva», *El Comercio*. http://www.elcomercio.com/actualidad/televisores-ensamblaje-precios-electrodomesticos-economia.html (accedido dic. 17, 2020).

[24] I. Malavolta, «Beyond native apps: web technologies to the rescue! (keynote)», en *Proceedings of the 1st International Workshop on Mobile Development - Mobile! 2016*, Amsterdam, Netherlands, 2016, pp. 1-2, doi: 10.1145/3001854.3001863.

[25] W. Jobe, «Native Apps Vs. Mobile Web Apps», *Int. J. Interact. Mob. Technol. IJIM*, vol. 7, n.º 4, p. 27, oct. 2013, doi: 10.3991/ijim.v7i4.3226.

[26] H. Heitkötter, S. Hanschke, y T. A. Majchrzak, «Evaluating Cross-Platform Development Approaches for Mobile Applications», en *Web Information Systems and Technologies*, vol. 140, J. Cordeiro y K.-H. Krempels, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 120-138.

[27] C. Gackenheimer, *Introduction to React*. Berkeley, CA: Apress, 2015.

[28] «React Native». https://reactnative.dev/ (accedido dic. 17, 2020).

[29] R. Giaffreda *et al.*, *Internet of Things. User-Centric IoT: First International Summit, IoT360 2014, Rome, Italy, October 27-28, 2014, Revised Selected Papers, Part I*. Springer, 2015.

[30] M. Yusufov y I. Kornilov, «Roles of Smart TV in IoT-environments: A survey», en *2013 13th Conference of Open Innovations Association (FRUCT)*, Petrozavodsk, abr. 2013, pp. 163-168, doi: 10.23919/FRUCT.2013.8124240.

[31] J. Holler, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, y D. Boyle, *Internet of Things: Introduction to a New Age of Intelligence*. San Diego, UNITED KINGDOM: Elsevier Science & Technology, 2014.

[32] A. D. Pathaka y J. V. Tembhurne, «Internet of Things: A Survey on IoT Protocols», *SSRN Electron. J.*, 2018, doi: 10.2139/ssrn.3168575.

[33] G. S. Karthick y P. B. Pankajavalli, «A Review on Human Healthcare Internet of Things: A Technical Perspective», *SN Comput. Sci.*, vol. 1, n.º 4, p. 198, jul. 2020, doi: 10.1007/s42979-020-00205-z.

[34] P. P. Ray, «A survey on Internet of Things architectures», *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 30, n.º 3, pp. 291-319, jul. 2018, doi: 10.1016/j.jksuci.2016.10.003.

[35] Z. Yang, Q. Zhou, L. Lei, K. Zheng, y W. Xiang, «An IoT-cloud Based Wearable ECG Monitoring System for Smart Healthcare», *J. Med. Syst.*, vol. 40, n.º 12, p. 286, dic. 2016, doi: 10.1007/s10916-016-0644-9.

[36] P. P. Ray, «A survey of IoT cloud platforms», *Future Comput. Inform. J.*, vol. 1, n.º 1-2, pp. 35-46, dic. 2016, doi: 10.1016/j.fcij.2017.02.001.

[37] M. Cardier *et al.*, «Telemedicina. Estado actual y perspectivas futuras en audiología y otología», *Rev. Médica Clínica Las Condes*, vol. 27, n.º 6, pp. 840-847, nov. 2016, doi: 10.1016/j.rmclc.2016.11.016.

[38] R. Zgheib, E. Conchon, y R. Bastide, «Semantic Middleware Architectures for IoT Healthcare Applications», en *Enhanced Living Environments: Algorithms, Architectures, Platforms, and Systems*, I. Ganchev, N. M. Garcia, C. Dobre, C. X. Mavromoustakis, y R. Goleva, Eds. Cham: Springer International Publishing, 2019, pp. 263-294.

[39] F. Stradolini, N. Tamburrano, T. Modoux, A. Tuoheti, D. Demarchi, y S. Carrara, «IoT for Telemedicine Practices enabled by an Android™ Application with Cloud System Integration», en *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, 2018, pp. 1-5, doi: 10.1109/ISCAS.2018.8351871.

[40] J. Wan *et al.*, «Wearable IoT enabled real-time health monitoring system», *EURASIP J. Wirel. Commun. Netw.*, vol. 2018, n.º 1, p. 298, dic. 2018, doi: 10.1186/s13638-018-1308-x.

[41] «Implementing pulse oximeter using MAX30100 - MORF - Coding And Engineering». https://morf.lv/implementing-pulse-oximeter-using-max30100 (accedido dic. 17, 2020).

[42] «Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health». maxim integrated, 09/14, [En línea]. Disponible en: https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf.

[43] F. Azhar, I. S. Ateeq, M. Haq, S. Shams, y A. Azhar, «An Hybrid Approach for Motion Artifact Elimination in Pulse Oximeter using MATLAB», abr. 2009.

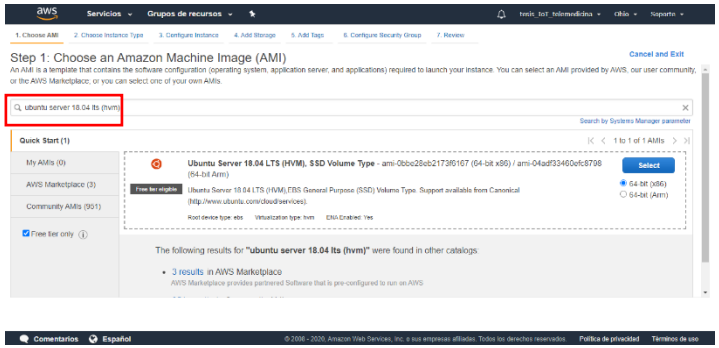
[44] «EC2 On-Demand Instance Pricing – Amazon Web Services». https://aws.amazon.com/es/ec2/pricing/on-demand/ (accedido dic. 17, 2020).

[45] «Biblioteca – Ministerio del Trabajo». http://www.trabajo.gob.ec/biblioteca/ (accedido dic. 17, 2020).

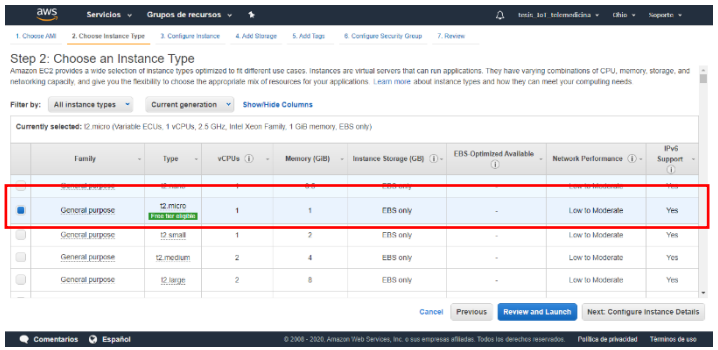
ANEXOS

ANEXO A: Instalación de la instancia en el servicio AWS

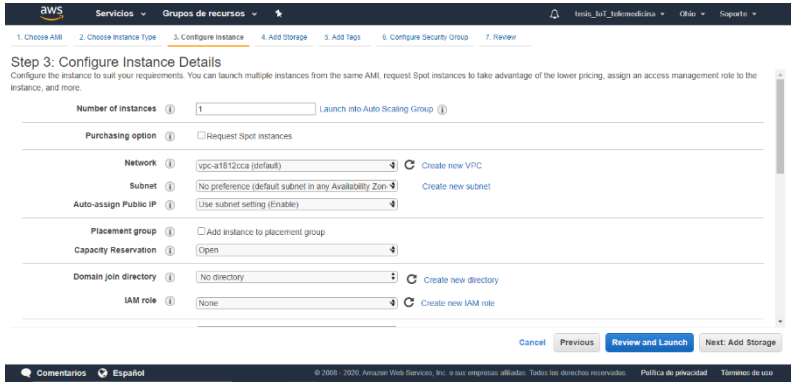
Seleccionar el tipo de máquina virtual.



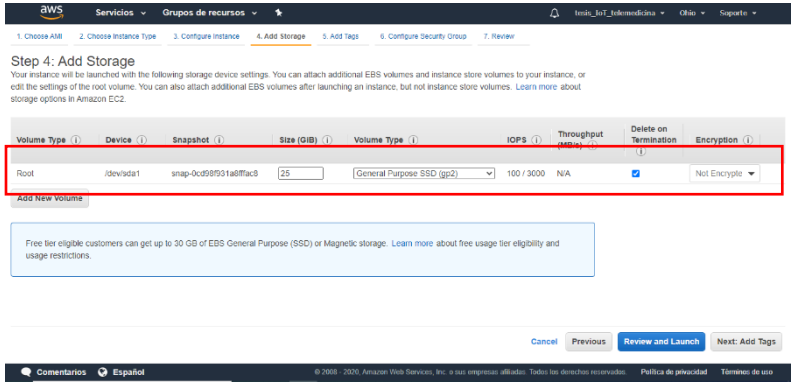
Seleccionar la instancia gratuita “t2.micro”



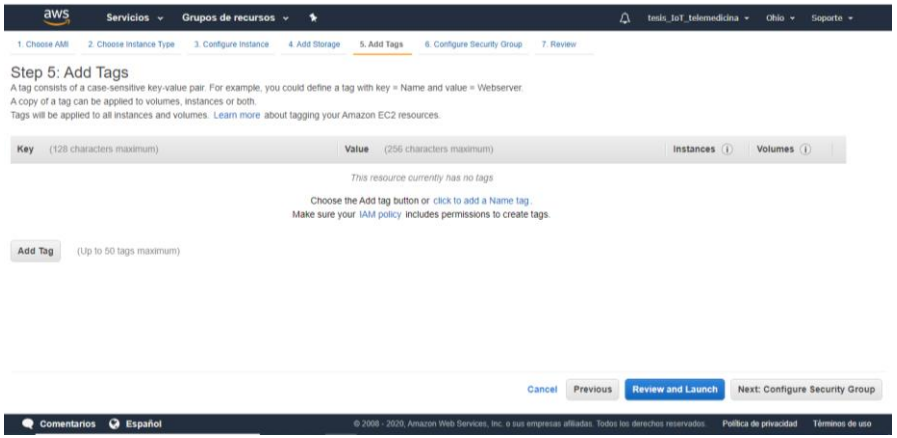
No modificar los detalles por defecto de la instancia



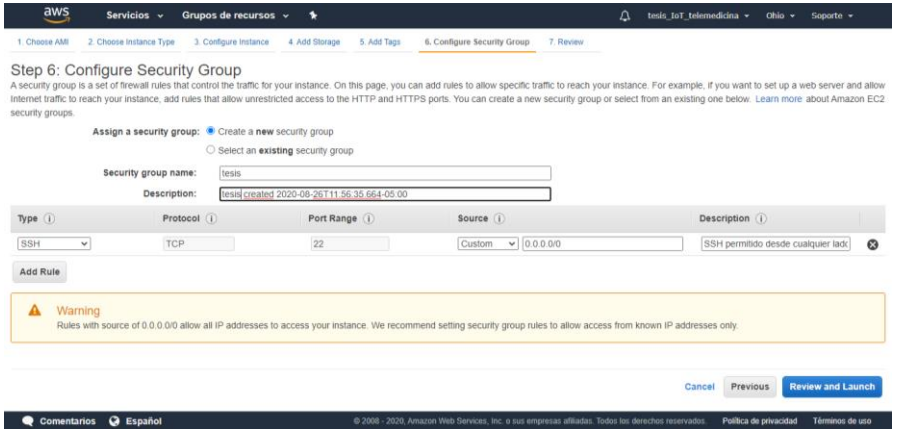
Seleccionar el disco de tipo SSD y el espacio de almacenamiento



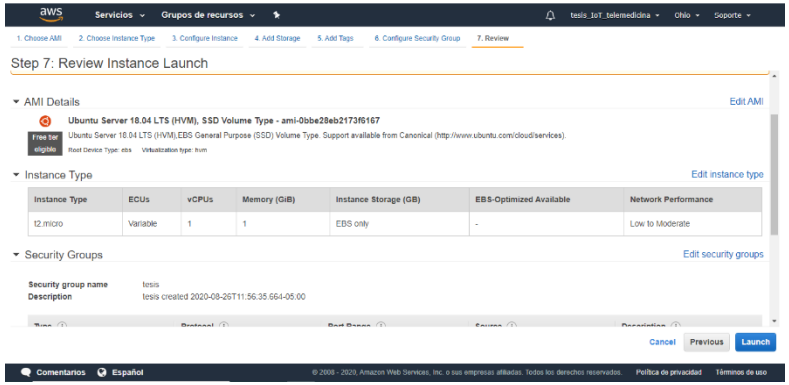
En caso de tener dos o más instancias agregar algún Tag afín al funcionamiento.



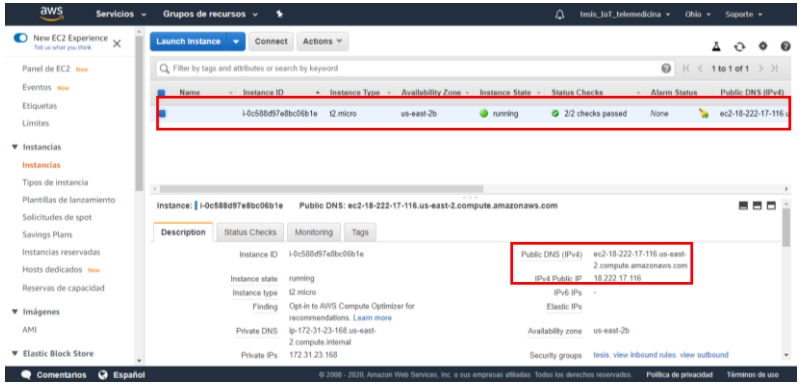
Creación de un grupo de seguridad para la conexión SSH y descargar el archivo “pem”.



Revisión de los detalles configurados para la instalación.

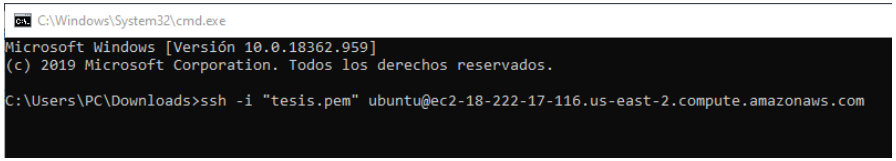


Una vez instalada, esta se ejecutará dando a conocer la IPV4 y el DNS públicos desde los cuales se puede acceder a la instancia de forma remota.

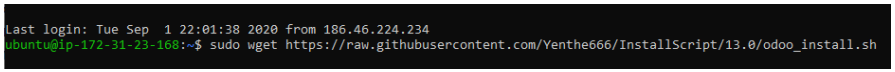


ANEXO B: Instalación de Odoo

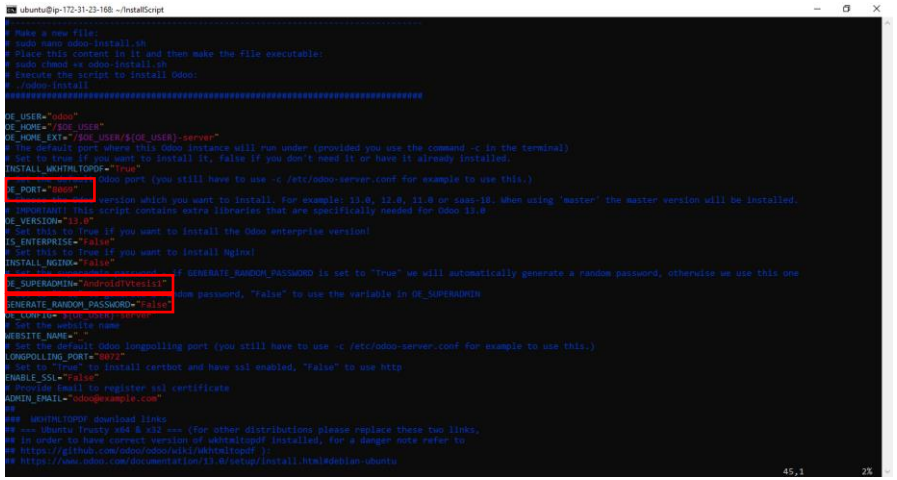
Abrir CMD y escribir "*ssh -i "tesis.pem" ubuntu@ ec2-18-222-17-116.us-east-2.compute.amazonaws.com"* para conectarse a la instancia por SSH.



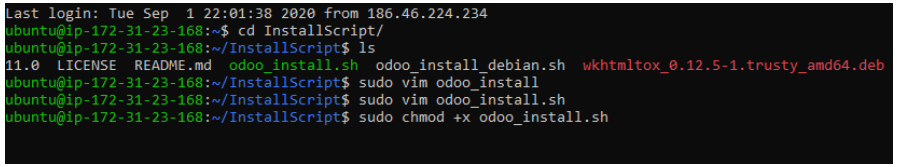
Descargar el archivo de instalación escribiendo el comando "*sudo wget https://raw.githubusercontent.com/Yenthe666/InstallScript/13.0/odoo_install.sh*".



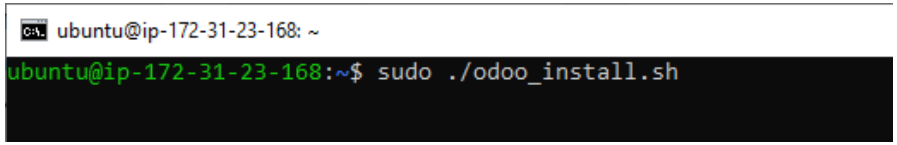
Modificar el archivo de instalación.



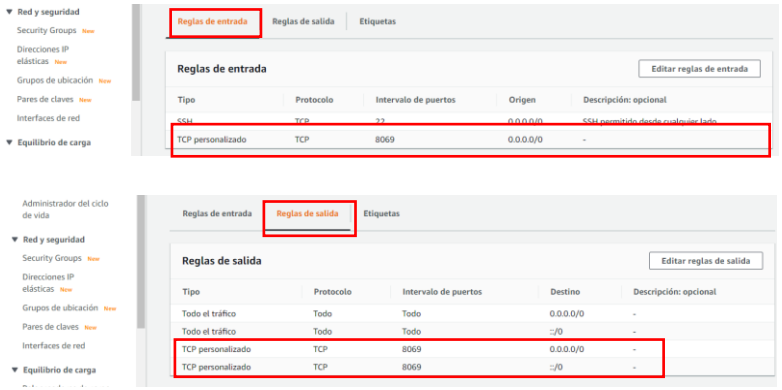
Dar permisos de ejecución al archivo de instalación con el comando "*sudo chmod +x odoo_install.sh*"



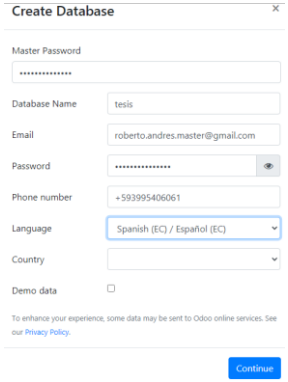
Ejecutar la instalación con el comando "*sudo ./odoo_install.sh*"



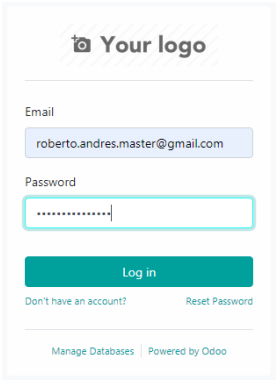
Habilitar el puerto 8069 tanto en transmisiones de entrada como de salida.



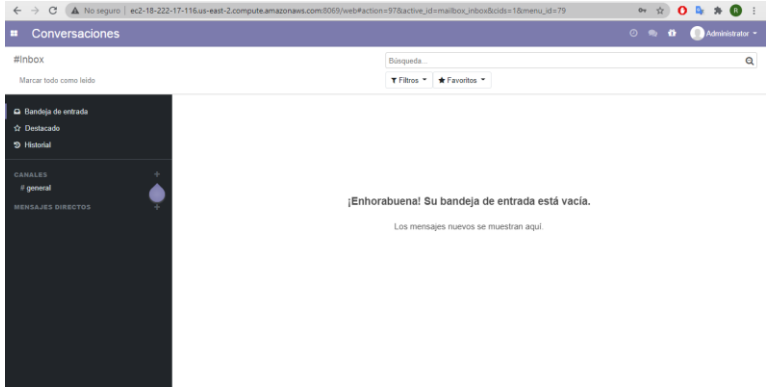
Dirigirse a la dirección IP publica seguido del puerto para configurar la base de datos.



Iniciar sesión dentro de Odoo con los datos ingresados en el paso anterior



Odoo está instalado y listo para su utilización



ANEXO C: Código fuente del archivo “models.py”

```python
from odoo import models, fields, api
from websocket import create_connection
import json
from base64 import b64encode


class ResPartner(models.Model):
    _inherit = 'res.partner'

    edad = fields.Integer(string='Edad')
    paciente = fields.Boolean(string='Paciente', default=False)
    res_paciente_ids = fields.One2many(
        'res.paciente', 'res_paciente', string='Detalles', copy=True)
    dispositivo = fields.Many2one(
        'tesis.dispositivos', string='Dispositivos', ondelete='cascade', index=True)

    def button_ver_grafica(self):
        return {
            'name': 'Signos Vitales',
            'view_type': 'form',
            'view_mode': 'graph',
            'res_model': 'res.paciente',
            'type': 'ir.actions.act_window',
            'context': {
                'search_default_res_paciente': self.id,
                'search_default_groupby_oxigeno': 1,
                'search_default_groupby_create_date': 1
            }
        }

    def button_prescripcion(self):
        return {
            'name': 'Prescripcion',
            'view_type': 'form',
            'view_mode': 'form',
            'res_model': 'tesis.prescripcion',
            'type': 'ir.actions.act_window',
            'context': {
                'default_res_paciente': self.id,
            }
        }
```

```python
class ResPaciente(models.Model):
    _name = "res.paciente"

    presion = fields.Integer(string='BPMs')
    oxigeno = fields.Integer(string='SpO2')
    res_paciente = fields.Many2one(
        'res.partner', string='Paciente', ondelete='cascade', index=True, domain="[('paciente', '=', True)]")


class Prescripcion(models.Model):
    _name = "tesis.prescripcion"

    mensaje = fields.Text(string='Mensaje')
    estado = fields.Selection([
        ('draft', 'Borrador'),
        ('send', 'Enviado')
    ], string='Estado', default='draft')
    mensaje_leido = fields.Boolean(string='Mensaje leido', default=False)
    notificacion = fields.Many2one(
        'tesis.notificaciones', string='Notificacion', ondelete='cascade', index=True, domain="[('atendido', '!=', True)]")
    res_paciente = fields.Many2one(
        'res.partner', string='Paciente', ondelete='cascade', index=True, domain="[('paciente', '=', True)]")
    res_user = fields.Many2one(
        'res.users', string='Usuario', readonly=True, default=lambda self: self.env.uid)

    def accion_enviar_mensaje(self):
        try:
            ws = create_connection("ws://18.222.17.116:80")
            data = {
                'tipo': 'notificacion',
                'titulo': 'Prescripción',
                'mensaje': 'Tienes una nueva prescripción médica',
                'prescripcion': self.id,
                'paciente': self.res_paciente.name,
                'dispositivo': self.res_paciente.dispositivo.codigo
            }
            if self.notificacion:
                self.notificacion.sudo().write({'atendido': True})

            result_encoded = b64encode(json.dumps(data).encode('utf-8'))
            ws.send(result_encoded.decode('utf-8'))
            ws.recv()
            ws.close()
            self.estado = 'send'
        except Exception as ex:
            print(ex)
            pass

    @api.onchange('res_paciente')
    def _onchange_paciente(self):
        domain = {'notificacion': [
            ('res_paciente.res_paciente', '=', self.res_paciente.id), ('atendido', '!=', True)]}
        return {'domain': domain}


class Dispositivos(models.Model):
    _name = "tesis.dispositivos"

    codigo = fields.Char(string='Codigo', required=True)
    res_paciente_ids = fields.One2many(
        'res.partner', 'dispositivo', string='Pacientes', copy=True)
```

```python
class Notificationes(models.Model):
    _name = "tesis.notificaciones"

    res_paciente = fields.Many2one(
        'res.paciente', string='Paciente Rel', ondelete='cascade', index=True)
    atendido = fields.Boolean(string='Atentido', default=False)
    name = fields.Char(string='Código', readonly=True, copy=False, default='/')
    nombre_paciente = fields.Char(
        string="Paciente", copy=False, related='res_paciente.res_paciente.name', readonly=True)
    presion = fields.Integer(string="BPMs", copy=False,
                             related='res_paciente.presion', readonly=True)
    oxigeno = fields.Integer(string="SpO2", copy=False,
                             related='res_paciente.oxigeno', readonly=True)

    @api.model
    def create(self, vals):
        vals['name'] = self.env['ir.sequence'].next_by_code(
            'sequence_tesis_notificaciones')

        return super(Notificationes, self).create(vals)

    def button_prescripcion(self):
        return {
            'name': 'Prescripcion',
            'view_type': 'form',
            'view_mode': 'form',
            'res_model': 'tesis.prescripcion',
            'type': 'ir.actions.act_window',
            'context': {
                'default_res_paciente': self.res_paciente.res_paciente.id,
                'default_notificacion': self.id,
            }
        }
```

ANEXO D: Código fuente del archivo “views.xml”

```
<odoo>
  <data>
    <record model="ir.ui.view" id="inherit_res_partner">
      <field name="name">res.partner.inherit</field>
      <field name="model">res.partner</field>
      <field name="inherit_id" ref="base.view_partner_form"/>
      <field name="arch" type="xml">
        <xpath expr="//field[@name='type']" position="before">
          <field name="edad" />
          <field name="paciente"/>
          <field name="dispositivo" />
        </xpath>

        <xpath expr="//sheet/div[@name='button_box']" position="inside">
          <button name="button_ver_grafica" type="object" class="oe_stat_button" icon="fa-area-chart">
            <span class="o_field_widget">Ver Gráfica</span>
          </button>

          <button name="button_prescripcion" type="object" class="oe_stat_button" icon="fa-envelope">
            <span class="o_field_widget">Prescripción</span>
          </button>
        </xpath>

        <xpath expr="//notebook" position="inside">
          <page string="Historial de signos Vitales">
            <field name="res_paciente_ids">
              <tree editable="bottom" default_order="create_date desc" decoration-danger="(presion&gt;100) or (presion&lt;60) or (oxigeno&lt;95)">
                <field name="create_date" string="Fecha" />
                <field name="presion"/>
                <field name="oxigeno"/>
              </tree>
            </field>
          </page>
        </xpath>
```

```
  <xpath expr="//notebook/page[1]" position="attributes">
    <attribute name="invisible">1</attribute>
  </xpath>

  <xpath expr="//notebook/page[2]" position="attributes">
    <attribute name="invisible">1</attribute>
  </xpath>

  <xpath expr="//notebook/page[3]" position="attributes">
    <attribute name="invisible">1</attribute>
  </xpath>

  <field name="function" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="type" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="phone" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="mobile" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="email" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="website" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="category_id" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>

  <field name="title" position="attributes">
    <attribute name="invisible">1</attribute>
  </field>


    <field name="vat" position="attributes">
      <attribute name="invisible">1</attribute>
    </field>


  </field>
</record>

<record model="ir.actions.act_window" id="tesis_res_partner_action_window">
  <field name="name">Pacientes</field>
  <field name="res_model">res.partner</field>
  <field name="domain">[("paciente", "=", True)]</field>
  <field name="view_mode">kanban,tree,form</field>
</record>


    <menuitem id="tesis_menu_root" groups="base.user_admin,group_monitor_tesis" name="ESalud" />

    <menuitem name="Pacientes" id="menu_root_tesis_res_partner" action="tesis_res_partner_action_window" parent="tesis_menu_root" groups="base.user_admin,group_monitor_tesis"

  </data>
</odoo>
```

ANEXO E: Código fuente del archivo “notificaciones.xml”

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <data>
        <record model="ir.ui.view" id="view_tesis_notificaciones_tree">
            <field name="name">tesis.notificaciones.tree</field>
            <field name="model">tesis.notificaciones</field>
            <field name="arch" type="xml">
                <tree decoration-danger="atendido == False" decoration-success="atendido == True">
                    <field name="name" />
                    <field name="nombre_paciente"/>
                    <field name="presion" />
                    <field name="oxigeno" />
                    <field name="create_date" string="Fecha" />
                    <field name="atendido"/>
                </tree>
            </field>
        </record>

        <record id="view_tesis_notificaciones_from" model="ir.ui.view">
            <field name="name">tesis.notificaciones.form</field>
            <field name="model">tesis.notificaciones</field>
            <field name="arch" type="xml">
                <form>
                    <sheet>
                        <div class="oe_button_box" name="button_box">
                            <button name="button_prescripcion" attrs="{'invisible': [('atendido', '=', True)]}" type="object" class="oe_stat_button" icon="fa-envelope">
                                <span class="o_field_widget">Antender Notificación</span>
                            </button>
                        </div>
                        <div class="oe_title">
                            <label for="name"/>
                            <h1>
                                <field name="name"/>
                            </h1>
                        </div>
                        <group col="2">
                            <group>
                                <field name="res_paciente" groups="base.user_admin" />
                                <field name="nombre_paciente" />
                                <field name="presion" />
                                <field name="oxigeno" />
                                <field name="atendido" readonly="1" />
                            </group>
                        </group>
                    </sheet>
                </form>
            </field>
        </record>


        <record id="action_tesis_notificaciones_open" model="ir.actions.act_window">
            <field name="name">Notificaciones</field>
            <field name="type">ir.actions.act_window</field>
            <field name="res_model">tesis.notificaciones</field>
            <field name="view_mode">tree,form</field>
        </record>

        <menuitem id="tesis_notificaciones_sub_menu" parent="tesis_menu_root" name="Notificaciones" action="action_tesis_notificaciones_open" groups="base.user_admin,group_mo

    </data>
</odoo>
```

ANEXO F: código fuente del archivo “prescripción.xml”

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <data>
        <record model="ir.ui.view" id="view_tesis_prescripcioc_tree">
            <field name="name">tesis.prescripcion.tree</field>
            <field name="model">tesis.prescripcion</field>
            <field name="arch" type="xml">
                <tree default_order="create_date desc" decoration-info="estado == 'draft'" decoration-success="estado == 'send'">
                    <field name="res_paciente"/>
                    <field name="create_date" string="Fecha" />
                    <field name="res_user" groups="base.user_admin" />
                    <field name="estado"/>
                    <field name="mensaje_leido" />
                </tree>
            </field>
        </record>

        <record id="view_tesis_prescripcion_from" model="ir.ui.view">
            <field name="name">tesis.prescripcion.form</field>
            <field name="model">tesis.prescripcion</field>
            <field name="arch" type="xml">
                <form>
                    <header>
                        <button name="accion_enviar_mensaje" attrs="{'invisible': [('estado', '=', 'send')]}" string="Enviar Mensaje" type="object" class="oe_highlight"/>
                        <field name="estado" widget="statusbar" statusbar_visible="draft,send"/>
                    </header>
                    <sheet>
                        <group col="2">
                            <group>
                                <field name="res_paciente" context="{'default_res_paciente': res_paciente}" attrs="{'readonly': [('estado', '=', 'send')]}" options="{'no_crea
                                <field name="res_user" groups="base.user_admin" />
                                <field name="mensaje_leido" readonly="1" />
                                <field name="notificacion" context="{'default_notificacion': notificacion}" attrs="{'readonly': [('estado', '=', 'send')]}" options="{'no_crea
                            </group>
                        </group>
                        <notebook>
                            <page string="Mensaje">
                                <field name="mensaje" nolabel="1" attrs="{'readonly': [('estado', '=', 'send')]}" />
                            </page>
                        </notebook>
                    </sheet>
                </form>
            </field>
        </record>

        <record id="action_tesis_prescripcio_open" model="ir.actions.act_window">
            <field name="name">Prescripciones</field>
            <field name="type">ir.actions.act_window</field>
            <field name="res_model">tesis.prescripcion</field>
            <field name="view_mode">tree,form</field>
        </record>

        <menuitem id="tesis_prescripcion_sub_menu" parent="tesis_menu_root" name="Prescripciones" action="action_tesis_prescripcio_open" groups="base.user_admin,group_monitor,

    </data>
</odoo>
```

ANEXO G: código fuente del archivo “dispositivos.xml”

```xml
<odoo>
    <data>
        <record model="ir.ui.view" id="view_tesis_dispositivos_tree">
            <field name="name">tesis.dispositivos.tree</field>
            <field name="model">tesis.dispositivos</field>
            <field name="arch" type="xml">
                <tree>
                    <field name="codigo"/>
                </tree>
            </field>
        </record>

        <record id="view_tesis_dispositivos_from" model="ir.ui.view">
            <field name="name">tesis.dispositivos.form</field>
            <field name="model">tesis.dispositivos</field>
            <field name="arch" type="xml">
                <form>
                    <sheet>
                        <group>
                            <field name="codigo" />
                        </group>
                        <notebook>
                            <page string="Pacientes">
                                <field name="res_paciente_ids" readonly="1">
                                    <tree>
                                        <field name="name" />
                                        <field name="edad" />
                                    </tree>
                                </field>
                            </page>
                        </notebook>
                    </sheet>
                </form>
            </field>
        </record>
        <record id="action_tesis_dispositivos_open" model="ir.actions.act_window">
            <field name="name">Dispositivos</field>
            <field name="type">ir.actions.act_window</field>
            <field name="res_model">tesis.dispositivos</field>
            <field name="view_mode">tree,form</field>
        </record>
        <menuitem id="dispositivos_sub_menu" parent="tesis_menu_root" name="Dispositivos" action="action_tesis_dispositivos_open" groups="base.user_admin" />
    </data>
</odoo>
```

ANEXO H: código fuente del archivo “controllers.py”

```
import json
import werkzeug
from odoo import http, SUPERUSER_ID
from odoo.http import request


class TesisController(http.Controller):
    def _response(self, response):
        mime = 'application/json; charset=utf-8'
        body = json.dumps(response)
        headers = [
            ('Content-Type', mime),
            ('Content-Length', len(body))
        ]
        return werkzeug.wrappers.Response(body, headers=headers)

    @http.route(['/api/datos/<string:idDispositivo>', '/api/paciente/<int:partner_id>'], auth='public', type='http', csrf=False, methods=["GET"], cors="*")
    def get_data(self, partner_id=False, idDispositivo=False, **kw):
        if partner_id:
            paciente = request.env["res.partner"].sudo().search(
                [('id', '=', partner_id)])
            paciente_result = {
                "id": paciente.id,
                "nombre": paciente.name,
                "edad": paciente.edad,
            }
            result = {'mensaje': 'Paciente recuperado',
                      'valor': paciente_result, 'success': True}
            return self._response(result)
        else:
            todos_pacientes = []
            dispositivo = request.env['tesis.dispositivos'].sudo().search(
                [('codigo', '=', idDispositivo)])
            pacientes = request.env["res.partner"].sudo().search(
                [('paciente', '=', True), ('dispositivo', '=', dispositivo.id), ('activo', '=', True)])
            for paciente in pacientes:
                todos_pacientes.append({
                    "id": paciente.id,
                    "nombre": paciente.name,
                    "edad": paciente.edad,
                })
            result = {'mensaje': 'Pacientes recuperados',
                      'valor': todos_pacientes, 'success': True}
            return self._response(result)
```

```python
@http.route(['/api/crear_usuario', '/api/modificar_usuario'], auth='public', type='json', csrf=False, methods=["POST", "PUT"])
def create(self, **kw):
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}
    if request.httprequest.method in ["PUT"]:
        values["activo"] = True
        paciente = request.env["res.partner"].sudo().search(
            [("id", "=", values["id"])])
        del values["id"]
        dispositivo = request.env['tesis.dispositivos'].sudo().search(
            [('codigo', '=', values.get('idDispositivo'))])
        values["dispositivo"] = dispositivo.id
        del values["idDispositivo"]
        paciente.write(values)
        nuevo_paciente_result = {
            "nombre": paciente.name,
            "edad": paciente.edad,
        }
        result = {'mensaje': 'Pacientes actualizado',
                  'paciente': nuevo_paciente_result, 'success': True}
        return result
    else:
        dispositivo = request.env['tesis.dispositivos'].sudo().search(
            [('codigo', '=', values.get('idDispositivo'))])
        values["dispositivo"] = dispositivo.id
        paciente = request.env["res.partner"].sudo().search(
            [("dispositivo", "=", dispositivo.id), ("name", "=", values.get("name"))])
        del values["idDispositivo"]
        if paciente:
            paciente.write({'activo': True})
            result = {'success': True}
            return result
        else:
            values["paciente"] = True
            values["activo"] = True
            pac = request.env["res.partner"].sudo().create(values)
            nuevo_paciente_result = {
                "nombre": pac.name,
                "edad": pac.edad,
            }
            result = {'mensaje': 'Pacientes creado',
                      'paciente': nuevo_paciente_result, 'success': True}
            return result
```

```python
@http.route('/api/borrar', auth='public', type='json', csrf=False, methods=["POST"])
def borrar(self, **kw):
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}
    request.env.uid = SUPERUSER_ID
    user = request.env["res.partner"].search([("id", "=", values["id"])])
    user.write({'activo': False})
    result = {'success': True}
    return result

@http.route('/api/splash', auth='public', type='json', csrf=False, methods=["POST"])
def splash_screen(self, **kw):
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}
    dispositivo = request.env['tesis.dispositivos'].sudo().search(
        [('codigo', '=', values.get('codigo'))])
    if not dispositivo:
        request.env['tesis.dispositivos'].sudo().create(values)
    result = {'success': True}
    return result

@http.route('/api/datosPaciente', auth='public', type='json', csrf=False, methods=["POST"])
def registrar_datos_paciente(self, **kw):
    request.session.authenticate(
        request.session.db, "roberto.andres.master@gmail.com", "AndroidTVtesis1")
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}
    paciente = request.env['res.paciente'].create(values)
    user = request.env["res.partner"].search(
        [("id", "=", values["res_paciente"])])
    if values.get('oxigeno') < 95 or values.get('presion') > 100 or values.get('presion') < 60:
        mensaje = 'El paciente '+user.name+", tiene los signos vitales. BPMs: " + \
            str(values.get("presion")) + ", SpO2: " + \
            str(values.get("oxigeno"))
        usuarios = request.env["res.users"].search([])
        for usuario in usuarios:
            usuario.notify_danger(message=mensaje)

        request.env['tesis.notificaciones'].create(
            {'res_paciente': paciente.id})
    result = {'success': True}
    return result

@http.route('/api/prescripcion/<int:idPaciente>', auth='public', type='http', csrf=False, methods=["GET"], cors="*")
def get_prescripcion(self, idPaciente=False, **kw):
    if idPaciente:
        prescripcion = request.env["tesis.prescripcion"].sudo().search(
            [("res_paciente", "=", idPaciente)], order='id desc', limit=1)
        if prescripcion:
            result = {'mensaje': prescripcion.mensaje, 'success': True}
        else:
            result = {
                "mensaje": "No se ha encontrado una prescripcion.", 'success': True}
        return self._response(result)
    else:
        return self._response({"mensaje": "No se ha encontrado una prescripcion.", 'success': True})
```

```python
@http.route('/api/grafica', auth='public', type='json', csrf=False, methods=["POST"])
def obtener_datos_grafica(self, **kw):
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}

    datos = request.env["res.paciente"].sudo().search([("res_paciente", "=", values.get("res_paciente")), (
        'create_date', '>=', values.get("start_date")), ('create_date', '<=', values.get("end_date"))])

    label = []
    datasets = [{'id': 1, 'data': []}, {'id': 2, 'data': []}]

    for dataset in datasets:
        for dato in datos:
            if dataset.get('id') == 1:
                dataset.get('data').append(dato.oxigeno)
            else:
                dataset.get('data').append(dato.presion)

    notificaciones = request.env["tesis.prescripcion"].sudo().search(
        [("res_paciente", "=", values.get("res_paciente")), (
            'create_date', '>=', values.get("start_date")), ('create_date', '<=', values.get("end_date"))])

    notis = []
    contador = 0

    for noti in notificaciones:
        notis.append({
            'id': noti.id,
            'mensaje': noti.mensaje,
            'fecha': noti.create_date.strftime('%d/%m/%Y'),
            'leido': noti.mensaje_leido
        })
        if not noti.mensaje_leido:
            contador = contador+1

    for info in datos:
        label.append(info.create_date.strftime('%d/%m/%Y'))

    return {'data': {'datasets': datasets, 'labels': label, 'legend': ['BPMs', 'SpO2']}, 'notificacion': {'datos': notis, 'contador': contador}}


@http.route('/api/actNoti', auth='public', type='json', csrf=False, methods=["PUT"])
def actualizar_notificaciones(self, **kw):
    values = request.httprequest.data and json.loads(
        request.httprequest.data.decode('utf-8')) or {}
    datos = request.env["tesis.prescripcion"].sudo().search(
        [("res_paciente", "=", values.get("res_paciente")), ('mensaje_leido', '!=', True)])
    for dato in datos:
        dato.write({'mensaje_leido': True})
    result = {'mensaje': 'Mensaje actualizado', 'success': True}
    return result
```

ANEXO I: Código fuente del archivo “MAX30100_BeatDetector.h”

```
#ifndef MAX30100_BEATDETECTOR_H
#define MAX30100_BEATDETECTOR_H

#include <stdint.h>

#define BEATDETECTOR_INIT_HOLDOFF               2000    // in ms, how long to wait before counting
#define BEATDETECTOR_MASKING_HOLDOFF            200     // in ms, non-retriggerable window after beat detection
#define BEATDETECTOR_BPFILTER_ALPHA             0.6     // EMA factor for the beat period value
#define BEATDETECTOR_MIN_THRESHOLD              20      // minimum threshold (filtered) value
#define BEATDETECTOR_MAX_THRESHOLD              800     // maximum threshold (filtered) value
#define BEATDETECTOR_STEP_RESILIENCY            30      // maximum negative jump that triggers the beat edge
#define BEATDETECTOR_THRESHOLD_FALLOFF_TARGET   0.3     // thr chasing factor of the max value when beat
#define BEATDETECTOR_THRESHOLD_DECAY_FACTOR     0.99    // thr chasing factor when no beat
#define BEATDETECTOR_INVALID_READOUT_DELAY      2000    // in ms, no-beat time to cause a reset
#define BEATDETECTOR_SAMPLES_PERIOD             10      // in ms, 1/Fs


typedef enum BeatDetectorState {
    BEATDETECTOR_STATE_INIT,
    BEATDETECTOR_STATE_WAITING,
    BEATDETECTOR_STATE_FOLLOWING_SLOPE,
    BEATDETECTOR_STATE_MAYBE_DETECTED,
    BEATDETECTOR_STATE_MASKING
} BeatDetectorState;


class BeatDetector
{
public:
    BeatDetector();
    bool addSample(float sample);
    float getRate();
    float getCurrentThreshold();

private:
    bool checkForBeat(float value);
    void decreaseThreshold();

    BeatDetectorState state;
    float threshold;
    float beatPeriod;
    float lastMaxValue;
    uint32_t tsLastBeat;
};

#endif
```

ANEXO J: Código fuente del archivo “MAX30100_BeatDetector.cpp”

```
#include <Arduino.h>

#include "MAX30100_BeatDetector.h"

#ifndef min
#define min(a,b) \
   ({ __typeof__ (a) _a = (a); \
       __typeof__ (b) _b = (b); \
     _a < _b ? _a : _b; })
#endif

BeatDetector::BeatDetector() :
    state(BEATDETECTOR_STATE_INIT),
    threshold(BEATDETECTOR_MIN_THRESHOLD),
    beatPeriod(0),
    lastMaxValue(0),
    tsLastBeat(0)
{
}

bool BeatDetector::addSample(float sample)
{
    return checkForBeat(sample);
}

float BeatDetector::getRate()
{
    if (beatPeriod != 0) {
        return 1 / beatPeriod * 1000 * 60;
    } else {
        return 0;
    }
}

float BeatDetector::getCurrentThreshold()
{
    return threshold;
}


 bool BeatDetector::checkForBeat(float sample)
 {
     bool beatDetected = false;

     switch (state) {
         case BEATDETECTOR_STATE_INIT:
             if (millis() > BEATDETECTOR_INIT_HOLDOFF) {
                 state = BEATDETECTOR_STATE_WAITING;
             }
             break;

         case BEATDETECTOR_STATE_WAITING:
             if (sample > threshold) {
                 threshold = min(sample, BEATDETECTOR_MAX_THRESHOLD);
                 state = BEATDETECTOR_STATE_FOLLOWING_SLOPE;
             }

             // Tracking lost, resetting
             if (millis() - tsLastBeat > BEATDETECTOR_INVALID_READOUT_DELAY) {
                 beatPeriod = 0;
                 lastMaxValue = 0;
             }

             decreaseThreshold();
             break;

         case BEATDETECTOR_STATE_FOLLOWING_SLOPE:
             if (sample < threshold) {
                 state = BEATDETECTOR_STATE_MAYBE_DETECTED;
             } else {
                 threshold = min(sample, BEATDETECTOR_MAX_THRESHOLD);
             }
             break;
```

```
        case BEATDETECTOR_STATE_MAYBE_DETECTED:
            if (sample + BEATDETECTOR_STEP_RESILIENCY < threshold) {
                // Found a beat
                beatDetected = true;
                lastMaxValue = sample;
                state = BEATDETECTOR_STATE_MASKING;
                float delta = millis() - tsLastBeat;
                if (delta) {
                    beatPeriod = BEATDETECTOR_BPFILTER_ALPHA * delta +
                            (1 - BEATDETECTOR_BPFILTER_ALPHA) * beatPeriod;
                }

                tsLastBeat = millis();
            } else {
                state = BEATDETECTOR_STATE_FOLLOWING_SLOPE;
            }
            break;

        case BEATDETECTOR_STATE_MASKING:
            if (millis() - tsLastBeat > BEATDETECTOR_MASKING_HOLDOFF) {
                state = BEATDETECTOR_STATE_WAITING;
            }
            decreaseThreshold();
            break;
    }

    return beatDetected;
}

void BeatDetector::decreaseThreshold()
{
    // When a valid beat rate readout is present, target the
    if (lastMaxValue > 0 && beatPeriod > 0) {
        threshold -= lastMaxValue * (1 - BEATDETECTOR_THRESHOLD_FALLOFF_TARGET) /
                (beatPeriod / BEATDETECTOR_SAMPLES_PERIOD);
    } else {
        // Asymptotic decay
        threshold *= BEATDETECTOR_THRESHOLD_DECAY_FACTOR;
    }

    if (threshold < BEATDETECTOR_MIN_THRESHOLD) {
        threshold = BEATDETECTOR_MIN_THRESHOLD;
    }
}
```

ANEXO K: Código fuente del archivo “MAX30100_SpO2Calculator.cpp”

```
#include <math.h>

#include "MAX30100_SpO2Calculator.h"

// SaO2 Look-up Table
// http://www.ti.com/lit/an/slaa274b/slaa274b.pdf
const uint8_t SpO2Calculator::spO2LUT[43] = {100,100,100,100,99,99,99,99,99,99,98,98,98,98,
                                             98,97,97,97,97,97,97,96,96,96,96,96,96,95,95,
                                             95,95,95,95,94,94,94,94,94,93,93,93,93,93};

SpO2Calculator::SpO2Calculator() :
    irACValueSqSum(0),
    redACValueSqSum(0),
    beatsDetectedNum(0),
    samplesRecorded(0),
    spO2(0)
{
}

void SpO2Calculator::update(float irACValue, float redACValue, bool beatDetected)
{
    irACValueSqSum += irACValue * irACValue;
    redACValueSqSum += redACValue * redACValue;
    ++samplesRecorded;

    if (beatDetected) {
        ++beatsDetectedNum;
        if (beatsDetectedNum == CALCULATE_EVERY_N_BEATS) {
            float acSqRatio = 100.0 * log(redACValueSqSum/samplesRecorded) / log(irACValueSqSum/samplesRecorded);
            uint8_t index = 0;

            if (acSqRatio > 66) {
                index = (uint8_t)acSqRatio - 66;
            } else if (acSqRatio > 50) {
                index = (uint8_t)acSqRatio - 50;
            }
            reset();

            spO2 = spO2LUT[index];
        }
    }
}


void SpO2Calculator::reset()
{
    samplesRecorded = 0;
    redACValueSqSum = 0;
    irACValueSqSum = 0;
    beatsDetectedNum = 0;
    spO2 = 0;
}

uint8_t SpO2Calculator::getSpO2()
{
    return spO2;
}
```

ANEXO L: Código fuente de la placa electrónica Arduino Mini Pro

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "MAX30100_PulseOximeter.h"
#include <avr/pgmspace.h>

const unsigned long intervaloEvento = 1000;
unsigned long tiempoAnterior = 0;

PulseOximeter pox;

struct Oximeter {
  unsigned int rate: 8;
  unsigned int spo: 7;
  unsigned int c: 6;
  unsigned int estado: 1;
};

Oximeter oxi;
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET 4
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

const char BPMS[] PROGMEM = "BPMs";  //0
const char SPO[] PROGMEM = "SpO2";  //1
const char ESPERA[] PROGMEM = "Esperando conexion";  //2
const char ESCOJER[] PROGMEM = "Usuario:";   //3

const char *const string_table[] PROGMEM = {BPMS, SPO, ESPERA, ESCOJER};
char buffer[5];

#define STATE 8
```

```
#define mostrar(dato,x,y){display.setCursor(x,y);display.print(dato);}
#define mostrargrafico(a,b,val,c,d){display.drawBitmap(a,b,val,c,d,WHITE);}

char* getStringfromProgMem(int i)
{
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table[i])));
  return buffer;
};

// '66%', 19x10px
const unsigned char seis [] PROGMEM = {
  0xff, 0xff, 0x80, 0x80, 0x00, 0x40, 0xbd, 0xe0, 0x40, 0xbd, 0xe0, 0x60, 0xbd, 0xe0, 0x60, 0xbd,
  0xe0, 0x60, 0xbd, 0xe0, 0x60, 0xbd, 0xe0, 0x40, 0x80, 0x00, 0x40, 0xff, 0xff, 0x80
};

// '33%', 19x10px
const unsigned char tres [] PROGMEM = {
  0xff, 0xff, 0x80, 0x80, 0x00, 0x40, 0xbc, 0x00, 0x40, 0xbc, 0x00, 0x60, 0xbc, 0x00, 0x60, 0xbc,
  0x00, 0x60, 0xbc, 0x00, 0x60, 0xbc, 0x00, 0x40, 0x80, 0x00, 0x40, 0xff, 0xff, 0x80
};

// '0%', 19x10px
const unsigned char cero [] PROGMEM = {
  0xff, 0xff, 0x80, 0x80, 0x00, 0x40, 0x80, 0x00, 0x40, 0x80, 0x00, 0x60, 0x80, 0x00, 0x60, 0x80,
  0x00, 0x60, 0x80, 0x00, 0x60, 0x80, 0x00, 0x40, 0x80, 0x00, 0x40, 0xff, 0xff, 0x80
};

// 'carga', 19x10px
const unsigned char carga [] PROGMEM = {
  0x7f, 0xff, 0x80, 0xff, 0xff, 0x80, 0xff, 0xff, 0xc0, 0xf3, 0x3f, 0xe0, 0xfc, 0x0f, 0xe0, 0xff,
  0x01, 0xe0, 0xff, 0xb1, 0xe0, 0xff, 0xff, 0xc0, 0xff, 0xff, 0x80, 0x7f, 0xff, 0x80
};

const unsigned char bluetooth [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xc0, 0x00, 0x03, 0xff, 0xf0, 0x00, 0x0f, 0xff, 0xfc, 0x00,
  0x1f, 0xff, 0xfe, 0x00, 0x3f, 0xff, 0xff, 0x00, 0x3f, 0xf7, 0xff, 0x00, 0x7f, 0xf3, 0xff, 0x80,
  0x7f, 0xf3, 0xff, 0x80, 0xff, 0xf1, 0xff, 0xc0, 0xff, 0xf0, 0xff, 0xc0, 0xff, 0xf0, 0x7f, 0xc0,
  0xff, 0xf2, 0x3f, 0xc0, 0xff, 0xf3, 0x1f, 0xc0, 0xfc, 0xf3, 0x8f, 0xc0, 0xfc, 0x73, 0x8f, 0xc0,
  0xfe, 0x33, 0x1f, 0xc0, 0xff, 0x12, 0x3f, 0xc0, 0xff, 0x80, 0x7f, 0xc0, 0xff, 0xc0, 0xff, 0xc0,
  0xff, 0xe1, 0xff, 0xc0, 0xff, 0xe1, 0xff, 0xc0, 0xff, 0xc0, 0xff, 0xc0, 0xff, 0x80, 0x7f, 0xc0,
  0xff, 0x12, 0x3f, 0xc0, 0xfe, 0x33, 0x1f, 0xc0, 0xfc, 0x73, 0x8f, 0xc0, 0xfe, 0xf3, 0x8f, 0xc0,
  0xff, 0xf3, 0x1f, 0xc0, 0xff, 0xf2, 0x3f, 0xc0, 0xff, 0xf2, 0x7f, 0xc0, 0xff, 0xf0, 0xff, 0xc0,
  0x7f, 0xf1, 0xff, 0xc0, 0x7f, 0xf1, 0xff, 0x80, 0x7f, 0xf3, 0xff, 0x80, 0x3f, 0xf7, 0xff, 0x00,
  0x1f, 0xff, 0xfe, 0x00, 0x0f, 0xff, 0xfc, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x03, 0xff, 0xf0, 0x00,
  0x00, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00
};

const unsigned char cora [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1c,
  0x00, 0x0f, 0x00, 0x00, 0x00, 0xff, 0x80, 0x7f, 0xc0, 0x00, 0x03, 0xff, 0xe1, 0xff, 0xf0, 0x00,
  0x07, 0xff, 0xf3, 0xff, 0xf8, 0x00, 0x0f, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x1f, 0xff, 0xff, 0xff,
  0xfe, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x3f, 0xff,
  0xff, 0xff, 0xff, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0x00,
  0x3f, 0xff, 0xff, 0xff, 0xff, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0x00, 0x3f, 0xff, 0xff, 0xff,
  0xff, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x1f, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x0f, 0xff,
  0xff, 0xff, 0xfc, 0x00, 0x07, 0xff, 0xff, 0xff, 0xf8, 0x00, 0x07, 0xff, 0xff, 0xff, 0xf8, 0x00,
  0x03, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x01, 0xff, 0xff, 0xff, 0xe0, 0x00, 0x00, 0xff, 0xff, 0xff,
  0xc0, 0x00, 0x00, 0x7f, 0xff, 0xff, 0x80, 0x00, 0x00, 0x3f, 0xff, 0xff, 0x00, 0x00, 0x00, 0x1f,
  0xff, 0xfe, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xfc, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x00,
  0x00, 0x03, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x01, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0xff, 0xc0,
  0x00, 0x00, 0x00, 0x00, 0x7f, 0x80, 0x00, 0x00, 0x00, 0x00, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x1e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

```
#define ANALOG 0
float outputValue = 0;
float smoothedVal = 0;
#define SAMPLES 15
#define VERF 4.5

static char palabra[15], *pSdata = palabra;
byte incomingByte;

void setup() {
  Serial.begin(9600);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.setTextColor(WHITE);
  display.clearDisplay();
  pinMode(STATE, INPUT);
  pinMode(3, OUTPUT);
  pox.begin();
}


void loop() {
  pox.update();
  outputValue = (VERF * analogRead(ANALOG)) / 1024;
  smoothedVal = smoothedVal + ((outputValue - smoothedVal) / SAMPLES);
  oxi.estado = digitalRead(STATE);
  oxi.estado ? (enlace()) : (noenlace());
}

void enlace() {
  display.clearDisplay();
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if ((char)incomingByte == 'T') {
      memset(palabra, 0, sizeof(palabra));
      pSdata = palabra;
      oxi.c = 0;
    } else {
      *pSdata++ = (char)incomingByte;
    }
  }
  unsigned long tiempoActual = millis();
  if (tiempoActual - tiempoAnterior >= intervaloEvento) {
    oxi.rate = pox.getHeartRate();
    oxi.spo = pox.getSpO2();
    if (oxi.rate && (char)incomingByte != 'T') {
      Serial.print(oxi.rate);
      Serial.print(',');
      Serial.print(oxi.spo);
      Serial.print(',');
      Serial.println(oxi.c);
      oxi.c++;
    }
    if (oxi.c == 60) {
      oxi.c = 0;
    }
    tiempoAnterior = tiempoActual;
  }
```

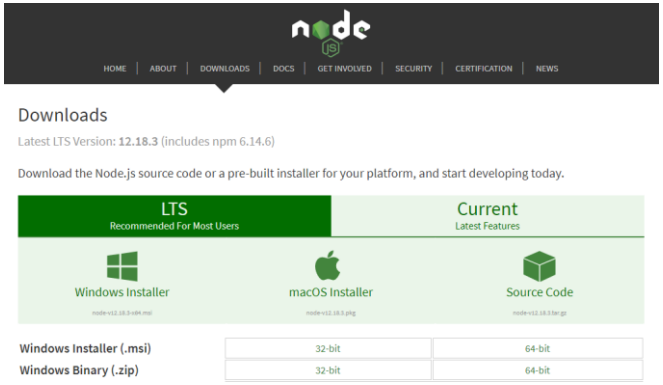
```
  mostrar(palabra, 0, 12);
  mostrargrafico(42, 20, cora, 42, 42);
  mostrar(oxi.rate, 15, 35);
  mostrar(oxi.spo, 105, 35);
  mostrar(getStringfromProgMem(0), 10, 52);
  mostrar(getStringfromProgMem(1), 100, 52);
  comparacion(smoothedVal);
  display.display();
}

void noenlace() {
  display.clearDisplay();
  unsigned long tiempoActual = millis();
  if (tiempoActual - tiempoAnterior >= intervaloEvento) {
    oxi.rate = pox.getHeartRate();
    oxi.spo = pox.getSpO2();
    tiempoAnterior = tiempoActual;
  }
  mostrar(getStringfromProgMem(2), 0, 12);
  mostrargrafico(52, 22, bluetooth, 26, 42);
  mostrar(oxi.rate, 15, 35);
  mostrar(oxi.spo, 105, 35);
  mostrar(getStringfromProgMem(0), 10, 52);
  mostrar(getStringfromProgMem(1), 100, 52);
  comparacion(smoothedVal);
  display.display();
}

void comparacion(float val) {
  if (val > 0.79) {
    mostrargrafico(105, 0, carga, 19, 10);
  }
  if (val > 0.73 && val < 0.79) {
    mostrargrafico(105, 0, seis, 19, 10);
  }
  if (val > 0.66 && val < 0.73) {
    mostrargrafico(105, 0, tres, 19, 10);
  }
  if (val < 0.66) {
    mostrargrafico(105, 0, cero, 19, 10);
  }
}
```

ANEXO M: instalación de las instancias

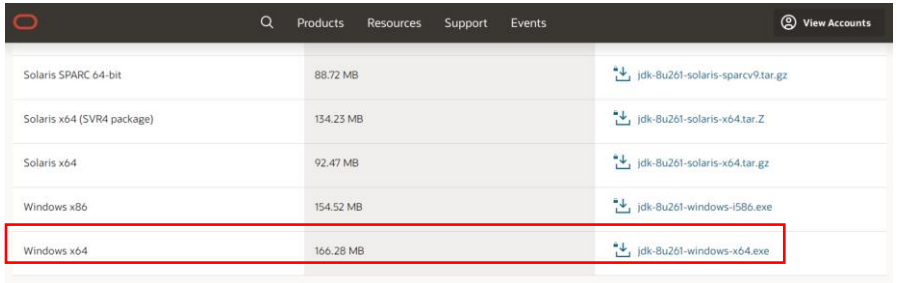
Para instalar NodeJS se debe descargar el instalador desde la página web.



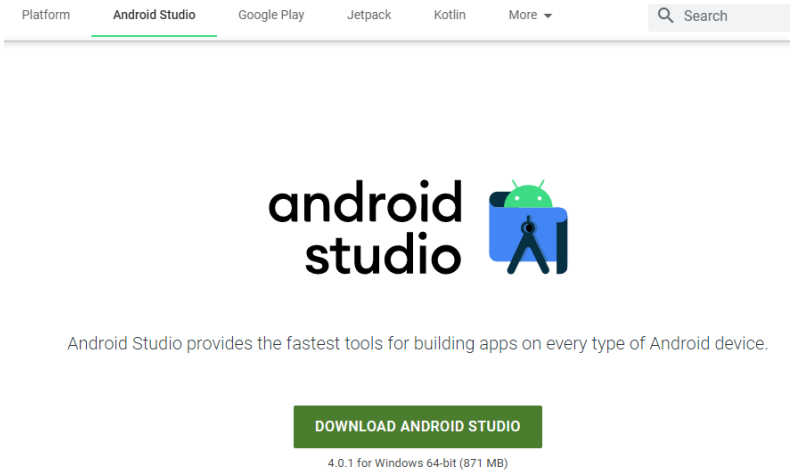
Para la instalación de Python se debe descargar el instalador desde la página web.

Se debe agregar al PATH el instalador para que el sistema lo reconozca como una variable.

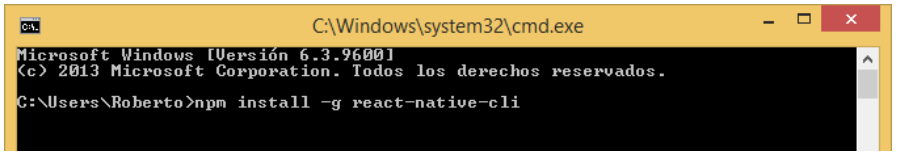
Se debe descarga e instalar el JDK desde la página de Oracle.



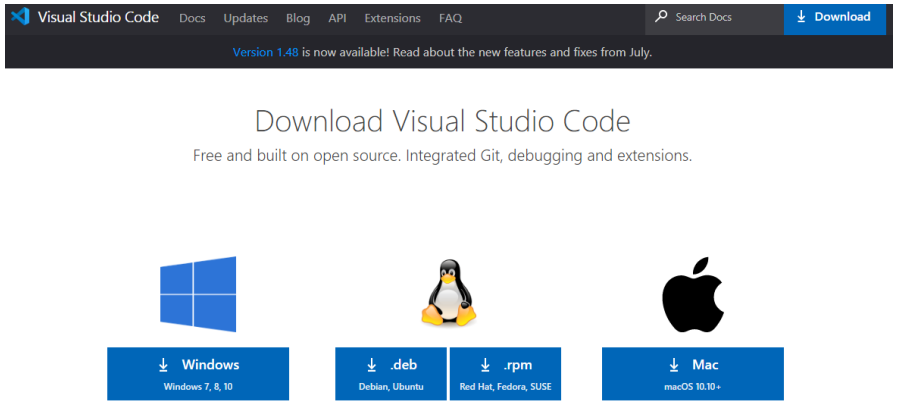
Para la instalación de Android Studio se debe descargar el instalador desde la página oficial de Android.



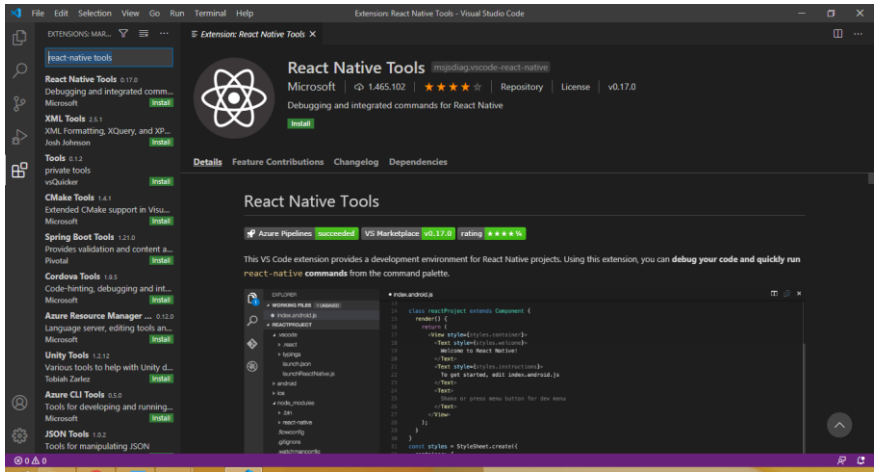
Para instalar el CLI de React-Native se debe escribir el comando "*npm install -g expo-cli*" desde un CMD.



Para la edición del código se instala Visual Studio Code que puede ser descargado desde su página web.



En visual Studio Code se instala "React Native Tools" para sintaxis y depuración del código.



ANEXO N: Código fuente del archivo “App.js”

```
import React, {useEffect, useRef, useState} from 'react';
import {NavigationContainer} from '@react-navigation/native';
import {createStackNavigator} from '@react-navigation/stack';
import {useNavigation} from '@react-navigation/native';
import {Alert, ToastAndroid, AppState} from 'react-native';
import NuevoPaciente from './CrearDatos';
import Datos from './DatosPaciente';
import Usuarios from './ListaUsuarios';
import Bluetooth from './ListaBluetooth';
import {AppProvider, useAppContext} from './Context';
import BluetoothSerial from 'react-native-bluetooth-serial-next';
import Splash from './SplashScreen';
import Sound from 'react-native-sound';
import DeviceInfo from 'react-native-device-info';
import Grafica from './Grafica'

var done = new Sound('done.mp3', Sound.MAIN_BUNDLE, (error) => {
  if (error) {
    console.log('failed to load the sound', error);
    return;
  }
});

const Lista = ({navigation}) => {
  const {isConected} = useAppContext();
  const {conected} = useAppContext();
  const appState = useRef(AppState.currentState);
  const [appStateVisible, setAppStateVisible] = useState(appState.current);

  useEffect(() => {
    const ws = new WebSocket('ws://18.222.17.116:8701', {
      transports: ['websocket'],
    });

    ws.onopen = () => {
      console.log('conectado');
    };
    ws.onerror = () => {
      console.log('error conectado');
    };
```

```
  ws.onmessage = (mensaje) => {
    const mensaje_decodificado = Buffer.from(mensaje.data, 'base64').toString(
      'ascii',
    );
    const mensaje_json = JSON.parse(mensaje_decodificado);
    if (mensaje_json.dispositivo === DeviceInfo.getUniqueId()) {
      done.play();
      ToastAndroid.showWithGravityAndOffset(
        mensaje_json.titulo +
          '\r\n' +
          '————————————————————————' +
          '\r\n' +
          'El paciente ' +
          ' e ' +
          mensaje_json.paciente +
          '\r\n' +
          mensaje_json.mensaje,
        ToastAndroid.LONG,
        ToastAndroid.TOP,
        500,
        0,
      );
    }
  };
}, []);

useEffect(() => {
  ActivacionAlerta();
}, []);

useEffect(() => {
  AppState.addEventListener('change', _handleAppStateChange);

  return () => {
    AppState.removeEventListener('change', _handleAppStateChange);
  };
}, []);
```

```
const _handleAppStateChange = (nextAppState) => {
  if (nextAppState === 'background') {
    setAppStateVisible(appState.current);
  }
  appState.current = nextAppState;
};

const ActivacionAlerta = () => {
  BluetoothSerial.on('connectionLost', ({device}) => {
    if (device && appState.current == 'active') {
      console.log('Presionado');
      Alert.alert(
        '¡Desconexión!',
        'La pulsera se ha desconectado/apagado. ¿Volver a conectarlo ahora?',
        [
          {
            text: 'NO',
            onPress: () => {
              isConected(false);
              BluetoothSerial.removeAllListeners = (eventName) =>
                DeviceEventEmitter.removeAllListeners(eventName);
            },
          },
          {
            text: 'SI',
            onPress: () => {
              navigation.navigate('Home');
              isConected(false);
              BluetoothSerial.removeAllListeners = (eventName) =>
                DeviceEventEmitter.removeAllListeners(eventName);
            },
          },
        ],
        {cancelable: false},
      );
    } else if (device && appState.current == 'background') {

        isConected(false);
        done.play();
        console.log('cambio');
        ToastAndroid.showWithGravityAndOffset(
          'Se ha perdido la conexión' +
            '\r\n' +
            '————————————————————————————' +
            '\r\n' +
            'La pulsera Weareable se ha apagado/desconectado' +
            '  .' +
            '\r\n' +
            'Vuelva a abrir la aplicación para conectarla.',
          ToastAndroid.LONG,
          ToastAndroid.TOP,
          500,
          0,
        );
      }
    });
  };
  return <Bluetooth />;
};
```

```
function App() {
  return (
    <AppProvider>
      <NavigationContainer>
        <Stack.Navigator initialRouteName="Splash">
          <Stack.Screen
            name="Home"
            component={Lista}
            options={{title: 'Bluetooth'}}
            options={{headerShown: false}}
          />
          <Stack.Screen
            name="Splash"
            component={Splash}
            options={{headerShown: false}}
          />
          <Stack.Screen
            name="Usuarios"
            component={Usuarios}
            options={{headerShown: false}}
          />
          <Stack.Screen
            name="Crear"
            component={NuevoPaciente}
            options={{headerShown: false}}
          />
          <Stack.Screen
            name="Datos"
            component={Datos}
            options={{headerShown: false}}
          />
          <Stack.Screen
            name="Grafica"
            component={Grafica}
            options={{headerShown: false}}
          />
        </Stack.Navigator>
      </NavigationContainer>
    </AppProvider>
  );
}

export default App;
```

ANEXO O: Código fuente de “SplashScreen.js”

```
import React, {useEffect, useState} from 'react';
import {View, Image,Alert} from 'react-native';
import {Dimensions} from 'react-native';
import {useNavigation} from '@react-navigation/native';
import {ActivityIndicator, Colors} from 'react-native-paper';
import DeviceInfo from 'react-native-device-info';

const Splash = () => {
  const windowWidth = Dimensions.get('window').width;
  const windowHeight = Dimensions.get('window').height;
  const navigation = useNavigation();

  useEffect(() => {
    setTimeout(async () => {
      await idDispositivo();
    }, 3000);
  }, []);

  const idDispositivo = async () => {
    const values = {
      codigo: DeviceInfo.getUniqueId(),
    };
    await fetch('http://18.222.17.116:8069/api/splash', {
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      method: 'POST',
      body: JSON.stringify(values),
    })

    .then((response) => response.json())
    .then((responseJson) => {
       if(responseJson.result.success){
         navigation.replace('Home');
       }
    })
    .catch((error) => {
      Alert.alert(
        '¡Error! ☹ Tiempo de espera agotado',
        'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
        [
          {
            text: 'OK',
          },
        ],
      );
      console.log(error);
    });
};
```

```
  return (
    <View style={{flex: 1, backgroundColor: 'white'}}>
      <View style={{flex: 1, justifyContent: 'center', alignSelf: 'stretch'}}>
        <Image
          style={{
            width: Math.round(windowWidth),
            height: Math.round(windowHeight),
            resizeMode: 'stretch',
          }}
          source={require('./splash.jpg')}
        />
      </View>
      <View
        style={{
          justifyContent: 'flex-end',
          alignSelf: 'center',
          marginBottom: 40,
        }}>
        <ActivityIndicator
          animating={true}
          color={Colors.red800}
          size={'large'}
        />
      </View>
    </View>
  );
};

export default Splash;
```

ANEXO P: Código fuente de “ListaBluetooth.js”

```
import React, {useState, useEffect} from 'react';
import {View, Text, StyleSheet, Image, ToastAndroid, Alert} from 'react-native';
import {
  Button,
  List,
  Switch,
  ActivityIndicator,
  Colors,
} from 'react-native-paper';
import {useNavigation} from '@react-navigation/native';
import BluetoothSerial from 'react-native-bluetooth-serial-next';
import {Buffer} from 'buffer';
import {useAppContext} from './Context';

global.Buffer = Buffer;

function Bluetooth() {
  const [dispositivos, setDispositivos] = useState([]);
  const [encender, setEncender] = useState(false);
  const [conectando, setConectando] = useState(false);
  const [texto, setTexto] = useState('Escoja pulsera');

  const navigate = useNavigation();
  const {setIdDevice} = useAppContext();
  const {isConected} = useAppContext();
  const {conected} = useAppContext();

  const onToggleSwitch = () => {
    if (!encender) {
      return habilitarBluetooth();
    }
    deshabilitarBluetooth();
  };

  const habilitarBluetooth = async () => {
    await BluetoothSerial.requestEnable();
    const lista = await BluetoothSerial.list();
    await BluetoothSerial.stopScanning();
    setDispositivos(lista);
    setEncender(true);
  };
```

```
const Emparejar = async (dispositivo) => {
  try {
    if (dispositivo.name === 'Weareable-Oximeter') {
      setConectando(true);
      setTexto('Esperando pulsera...');

      const conectado = await BluetoothSerial.device(
        dispositivo.id,
      ).connect();

      if (conectado) {
        ToastAndroid.show('Conectado', ToastAndroid.SHORT);
        setConectando(false);
        setTexto('Conectado');
        isConected(true);
        setIdDevice(dispositivo.id);
        navigate.navigate('Usuarios');
      } else {
        ToastAndroid.show(
          'Hubo un problema en la conexion',
          ToastAndroid.SHORT,
        );
        setConectando(false);
      }
    } else {
      ToastAndroid.show('No es un dispositivo valido', ToastAndroid.SHORT);
      setConectando(false);
    }
  } catch (e) {
    setConectando(false);
    Alert.alert(
      'Tiempo de espera agotado',
      'Presione OK para volver a intentarlo',
    );
    ToastAndroid.show('Error de conexión', ToastAndroid.SHORT);
    console.log(e);
  }
};
```

```
const Mapeo = () => {
  if (conectando) {
    return (
      <View style={{alignItems: 'center', justifyContent: 'center'}}>
        <ActivityIndicator
          animating={true}
          color={Colors.red800}
          size={'large'}
        />
      </View>
    );
  } else {
    return dispositivos.map((val, index) => (
      <View key={index} style={{padding: 10}}>
        <Button
          color="#C1CBF4"
          compact={true}
          mode="contained"
          onPress={() => {
            Emparejar(val);
          }}>
          <Text>{val.name}</Text>
        </Button>
        <Separator />
      </View>
    ));
  }
};

const Separator = () => <View style={styles.separator} />;


const Encender = () => {
  if (conectando) {
    return <Text>Intentando contectar...</Text>;
  } else {
    return (
      <View
        style={{
          justifyContent: 'center',
          alignItems: 'center',
        }}>
        <Switch
          color="#E82121"
          value={encender}
          onValueChange={onToggleSwitch}
        />
      </View>
    );
  }
};
```

```
const Emparejado = () => {
  if (conected) {
    return (
      <View style={styles.container}>
        <Text style={{fontSize: 30}}>Ya estas emparejado!</Text>
        <Separator />
        <Text style={{fontSize:15}}>Apagar bluetooth.</Text>
        <Switch
          color="#E82121"
          value={encender}
          onValueChange={onToggleSwitch}
        />
        <Button
          color="#C1CBF4"
          compact={true}
          mode="contained"
          onPress={() => navigate.navigate('Usuarios')}>
          <Text>Regresar</Text>
        </Button>
      </View>
    );
  } else {
    return (
      <View style={styles.container}>
        <Text>Bluetooth Encendido</Text>
        <Encender />
        <List.Section
          style={{
            margin: 10,
            borderWidth: 0.5,
            backgroundColor: '#F1EFEF',
            borderTopLeftRadius: 10,
            borderTopRightRadius: 10,
            borderBottomLeftRadius: 10,
            borderBottomRightRadius: 10,
          }}>
          <List.Subheader style={{fontSize: 20}}>{texto}</List.Subheader>
          <Mapeo />
        </List.Section>
      </View>
    );
  }
};
```

```
  return !encender ? (
    <View
      style={{
        justifyContent: 'center',
        alignItems: 'center',
        flex: 1,
      }}>
      <Text>Encender Bluetooth</Text>
      <Encender />

      <Image
        source={require('./android.png')}
        style={{
          height: 250,
          width: 250,
          resizeMode: 'contain',
        }}></Image>
      <Button
        color="#C1CBF4"
        compact={true}
        mode="contained"
        onPress={() => navigate.navigate('Usuarios')}>
        <Text>Continuar sin conecar la pulsera</Text>
      </Button>
    </View>
  ) : (
    <Emparejado />
  );
}
```

```
const styles = StyleSheet.create({
  scrollView: {
    marginHorizontal: 20,
  },
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    textAlign: 'center',
    marginVertical: 8,
    fontSize: 30,
  },
  fixToText: {
    flexDirection: 'row',
    justifyContent: 'space-between',
  },
  separator: {
    marginVertical: 8,
    borderBottomColor: '#737373',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
  containerValue: {
    backgroundColor: '#ff7550',
    borderRadius: 8,
    elevation: 6,
  },
});

export default Bluetooth;
```

ANEXO Q: Código fuente de “ListaUsuarios.js”

```
import React, {useEffect, useState} from 'react';
import {
  View,
  Text,
  StyleSheet,
  ScrollView,
  Image,
  Alert,
  ToastAndroid,
} from 'react-native';
import {
  Provider as PaperProvider,
  ActivityIndicator,
  Colors,
  Button,
  List,
  DefaultTheme,
  configureFonts,
} from 'react-native-paper';
import {useNavigation} from '@react-navigation/native';
import BluetoothSerial from 'react-native-bluetooth-serial-next';
import {Buffer} from 'buffer';
import {useAppContext} from './Context';
import DeviceInfo from 'react-native-device-info';

const abortController = new AbortController();
const signal = abortController.signal;
let isMounted = false;
```

```
const Usuarios = ({navigation}) => {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const navigate = useNavigation();
  const {idDevice} = useAppContext();

  const getData = async () => {
    write(idDevice, 'T');
    isMounted = true;
    await fetch(
      'http://192.168.1.11:8069/api/datos/' + DeviceInfo.getUniqueId(),
      {
        headers: {
          Accept: 'application/json',
        },
        signal: signal,
        method: 'GET',
      },
    )
      .then((response) => response.json())
      .then((responseJson) => {
        if (isMounted) {
          setData(responseJson.valor);
          setLoading(false);
        }
      })
      .catch((error) => {
        if (error) {
          console.log(error);
          Alert.alert(
            '¡Error! 😕 Tiempo de espera agotado',
            'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
            [
              {
                text: 'OK',
              },
            ],
          );
        }
      });
  };
```

```
useEffect(() => {
  const unsubscribe = navigate.addListener('focus', async () => {
    await getData();

  });
  //write(idDevice, 'T');
  return unsubscribe;
}, [navigate]);

const Separator = () => <View style={styles.separator} />;

const write = (id, data) => {
  if (typeof data === 'string') {
    const qq = new Buffer(data);
    console.log(Buffer.from(data).toString('base64'));
  }
  return BluetoothSerial.writeToDevice(
    Buffer.from(data).toString('base64'),
    id,
  );
};

const Mapeo = () => {
  return data.map((val, index) => (
    <View key={index} style={{padding: 10}}>
      <Button
        color="#C1CBF4"
        compact={true}
        mode="contained"
        onPress={() => navigate.navigate('Datos', val)}>
        <Text>{val.nombre}</Text>
      </Button>
      <Separator />
    </View>
  ));
};
```

```
const theme = {
  roundness: 2,
  ...DefaultTheme,
  fonts: configureFonts(fontConfig),
  ...DefaultTheme.colors,
  colors: {
    primary: '#3498db',
    accent: '#f1c40f',
  },
};
const fontConfig = {
  default: {
    regular: {
      fontFamily: 'sans-serif',
      fontWeight: 'normal',
    },
    medium: {
      fontFamily: 'sans-serif-medium',
      fontWeight: 'normal',
    },
    light: {
      fontFamily: 'sans-serif-light',
      fontWeight: 'normal',
    },
    thin: {
      fontFamily: 'sans-serif-thin',
      fontWeight: 'normal',
    },
  },
};
```

```
return loading ? (
  <PaperProvider theme={theme}>
    <Image
      style={{resizeMode: 'stretch', width: 1000, height: 120}}
      source={{
        uri: 'https://www.uta.edu.ec/v3.2/uta/images/header.png',
      }}
    />
    <View style={{flex: 1, justifyContent: 'center'}}>
      <ActivityIndicator
        animating={true}
        color={Colors.red800}
        size={'large'}
      />
    </View>
  </PaperProvider>
) : (
  <View>
    <View>
      <Image
        style={{resizeMode: 'stretch', width: 1000, height: 120}}
        source={{
          uri: 'https://www.uta.edu.ec/v3.2/uta/images/header.png',
        }}
      />
    </View>
```

```
      <View style={{height: 300}}>
        <ScrollView style={styles.scrollView}>
          <List.Section
            style={{
              margin: 100,
              borderWidth: 0.5,
              backgroundColor: '#F1EFEF',
              borderTopLeftRadius: 10,
              borderTopRightRadius: 10,
              borderBottomLeftRadius: 10,
              borderBottomRightRadius: 10,
            }}>
            <List.Subheader style={{fontSize: 20}}>
              Seleccione su usuario.
            </List.Subheader>
            <Mapeo />
          </List.Section>
        </ScrollView>
      </View>
      <View style={{alignSelf: 'center'}}>
        <Button
          compact={true}
          mode="contained"
          color="#C1CBF4"
          onPress={() => navigation.navigate('Crear', (id = {}))}>
          <Text>Crear usuario</Text>
        </Button>
      </View>
    </View>
  );
};
```

```
const styles = StyleSheet.create({
  scrollView: {
    marginHorizontal: 20,
  },
  container: {
    flex: 1,
    justifyContent: 'center',
    marginHorizontal: 16,
  },
  title: {
    textAlign: 'center',
    marginVertical: 8,
    fontSize: 30,
  },
  fixToText: {
    flexDirection: 'row',
    justifyContent: 'space-between',
  },
  separator: {
    marginVertical: 8,
    borderBottomColor: '#737373',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
  containerValue: {
    backgroundColor: '#ff7550',
    borderRadius: 8,
    elevation: 6,
  },
});

export default Usuarios;
```

ANEXO R: Código fuente del archivo “DatosPaciente.js”

```
import React, {useEffect, useState} from 'react';
import {
  TextInput,
  Card,
  Avatar,
  Button,
  Text,
  HelperText,
} from 'react-native-paper';
import {StyleSheet, View, Alert} from 'react-native';
import TextInputMask from 'react-native-text-input-mask';
import {validate} from './validate';
import {useNavigation} from '@react-navigation/native';
import DeviceInfo from 'react-native-device-info';

const NuevoPaciente = ({route}) => {
  const navigation = useNavigation();
  const [datos, setDatos] = useState({
    edad: '',
    edadError: '',
    nombre: '',
    nombreError: '',
    boton: 'Crear',
    mensaje: 'Usuario Creado',
    borrar: false,
  });

  useEffect(() => {
    const Editar = () => {
      if (route.params.id) {
        setDatos({
          edad: route.params.edad,
          edadError: '',
          nombre: route.params.nombre,
          nombreError: '',
          boton: 'Modificar',
          mensaje: 'Usuario Modificado',
          borrar: true,
        });
      }
    };
    Editar();
  }, []);
```

```
const constraints = {
  edad: {
    presence: {
      allowEmpty: false,
      message: 'La edad no debe estar vacia',
    },
  },
  nombre: {
    presence: {
      allowEmpty: false,
      message: 'El nombre no debe estar vacio',
    },
  },
};

const validateFields = (tipo, valor) => validate(tipo, valor, constraints);

const crearPaciente = async () => {
  let id = {};
  if (route.params.id) {
    id = {id: route.params.id};
  }
  const values = {
    ...id,
    name: datos.nombre,
    edad: datos.edad,
    idDispositivo: DeviceInfo.getUniqueId(),
  };
  await fetch(
    !route.params.id
      ? 'http://192.168.1.11:8069/api/crear_usuario/'
      : 'http://192.168.1.11:8069/api/modificar_usuario',
    {
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      method: !route.params.id ? 'POST' : 'PUT',
      body: JSON.stringify(values),
    },
  )
    .then((response) => response.json())
    .then((responseJson) => {
      !route.params.id
        ? navigation.navigate('Usuarios')
        : navigation.navigate('Datos');
    })
```

```
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! ☹ Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};

const User = (props) => (
  <Avatar.Image
    size={48}
    source={{
      uri:
        'https://www.psybooks.com/wp-content/uploads/psybooks-user-accounts.png',
    }}
  />
);
const Medico = (props) => (
  <Avatar.Image
    size={48}
    source={{
      uri:
        'https://image.freepik.com/vector-gratis/ilustracion-clinica-doctor_1270-69.jpg',
    }}
  />
);

const borrarUsuario = async () => {
  await fetch('http://192.168.1.11:8069/api/borrar', {
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    method: 'POST',
    body: JSON.stringify({id: route.params.id}),
  })
    .then((response) => response.json())
    .then((responseJson) => {
      navigation.navigate('Usuarios');
    })
```

```
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! ☹ Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};

const Borrar = (props) =>
  datos.borrar ? (
    <Button
      compact={true}
      color="#5AADF1"
      mode="contained"
      onPress={() =>
        Alert.alert(
          'Alerta',
          'Desde eliminar este usuario?',
          [
            {
              text: 'Cancel',
              style: 'cancel',
            },
            {text: 'Aceptar', onPress: async () => await borrarUsuario()},
          ],
          {cancelable: false},
        )
      }>
      <Text>Borrar usuario</Text>
    </Button>
  ) : (
    <Text></Text>
  );

const Separator = () => <View style={styles.separator} />;
```

```
return (
  <View style={{flexDirection: 'column'}}>
    <View style={{flexDirection: 'row'}}>
      <Card
        style={{
          flex: 1,
          padding: 50,
        }}>
        <Card.Title
          title="Ingrese sus datos"
          left={User}
          subtitle="Seleccione los campos con el puntero para poder modificarlos."
        />
        <TextInput
          render={(props) => <TextInputMask {...props} mask="[99]" />}
          style={{backgroundColor: 'transparent'}}
          placeholder="Ingrese su edad"
          maxLength={2}
          label="Edad"
          mode={'outlined'}
          error={datos.edadError !== '' && datos.edadError !== null}
          underlineColor="blue"
          value={datos.edad}
          onChangeText={(text) =>
            setDatos({
              ...datos,
              edad: text,
              edadError: validateFields('edad', text),
            })
          }
        />
        <HelperText
          type="error"
          visible={datos.edadError !== '' && datos.edadError !== null}>
          {datos.edadError}
        </HelperText>
        <TextInput
          placeholder="Ingrese su nombre"
          style={{backgroundColor: 'transparent'}}
          label="Nombre paciente"
          mode={'outlined'}
          error={datos.nombreError !== '' && datos.nombreError !== null}
          underlineColor="blue"
          value={datos.nombre}
          onChangeText={(text) =>
            setDatos({
              ...datos,
              nombre: text,
              nombreError: validateFields('nombre', text),
```

```
            })
          }
        />
        <HelperText
          type="error"
          visible={datos.nombreError !== '' && datos.nombreError !== null}>
          {datos.nombreError}
        </HelperText>
      </Card>
    </View>
    <Button
      compact={true}
      color="#5AADF1"
      mode="contained"
      onPress={crearPaciente}>
      <Text>{datos.boton}</Text>
    </Button>
    <Separator />
    <Borrar />
  </View>
 );
};

const styles = StyleSheet.create({
  separator: {
    marginVertical: 8,
    borderBottomColor: '#737373',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
});

export default NuevoPaciente;
```

ANEXO S: Código fuente del archivo “CrearDatos.js”

```
import React, {useEffect, useState} from 'react';
import {
  TextInput,
  Card,
  Avatar,
  Button,
  Text,
  HelperText,
} from 'react-native-paper';
import {StyleSheet, View, Alert} from 'react-native';
import TextInputMask from 'react-native-text-input-mask';
import {validate} from './validate';
import {useNavigation} from '@react-navigation/native';
import DeviceInfo from 'react-native-device-info';

const NuevoPaciente = ({route}) => {
  const navigation = useNavigation();
  const [datos, setDatos] = useState({
    edad: '',
    edadError: '',
    nombre: '',
    nombreError: '',
    boton: 'Crear',
    mensaje: 'Usuario Creado',
    borrar: false,
  });

  useEffect(() => {
    const Editar = () => {
      if (route.params.id) {
        setDatos({
          edad: route.params.edad,
          edadError: '',
          nombre: route.params.nombre,
          nombreError: '',
          boton: 'Modificar',
          mensaje: 'Usuario Modificado',
          borrar: true,
        });
      }
    };
    Editar();
```

```
const constraints = {
  edad: {
    presence: {
      allowEmpty: false,
      message: 'La edad no debe estar vacia',
    },
  },
  nombre: {
    presence: {
      allowEmpty: false,
      message: 'El nombre no debe estar vacio',
    },
  },
};

const validateFields = (tipo, valor) => validate(tipo, valor, constraints);

const crearPaciente = async () => {
  let id = {};
  if (route.params.id) {
    id = {id: route.params.id};
  }
  const values = {
    ...id,
    name: datos.nombre,
    edad: datos.edad,
    idDispositivo: DeviceInfo.getUniqueId(),
  };
  await fetch(
    !route.params.id
      ? 'http://192.168.1.11:8069/api/crear_usuario/'
      : 'http://192.168.1.11:8069/api/modificar_usuario',
    {
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      method: !route.params.id ? 'POST' : 'PUT',
      body: JSON.stringify(values),
    },
  )
```

```
    .then((response) => response.json())
    .then((responseJson) => {
      !route.params.id
        ? navigation.navigate('Usuarios')
        : navigation.navigate('Datos');
    })
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! 😕 Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};

const User = (props) => (
  <Avatar.Image
    size={40}
    source={{
      uri:
        'https://www.psybooks.com/wp-content/uploads/psybooks-user-accounts.png',
    }}
  />
);
const Medico = (props) => (
  <Avatar.Image
    size={40}
    source={{
      uri:
        'https://image.freepik.com/vector-gratis/ilustracion-clinica-doctor_1270-69.jpg',
    }}
  />
);
```

```
const borrarUsuario = async () => {
  await fetch('http://192.168.1.11:8069/api/borrar', {
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    method: 'POST',
    body: JSON.stringify({id: route.params.id}),
  })
    .then((response) => response.json())
    .then((responseJson) => {
      navigation.navigate('Usuarios');
    })
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! 🙁 Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};

 const Borrar = (props) =>
   datos.borrar ? (
     <Button
       compact={true}
       color="#5AADF1"
       mode="contained"
       onPress={() =>
         Alert.alert(
           'Alerta',
           'Desde eliminar este usuario?',
           [
             {
               text: 'Cancel',
               style: 'cancel',
             },
             {text: 'Aceptar', onPress: async () => await borrarUsuario()},
           ],
           {cancelable: false},
         )
       }>
       <Text>Borrar usuario</Text>
     </Button>
   ) : (
     <Text></Text>
   );

 const Separator = () => <View style={styles.separator} />;
```

```
return (
  <View style={{flexDirection: 'column'}}>
    <View style={{flexDirection: 'row'}}>
      <Card
        style={{
          flex: 1,
          padding: 50,
        }}>
        <Card.Title
          title="Ingrese sus datos"
          left={User}
          subtitle="Seleccione los campos con el puntero para poder modificarlos."
        />
        <TextInput
          render={(props) => <TextInputMask {...props} mask="[99]" />}
          style={{backgroundColor: 'transparent'}}
          placeholder="Ingrese su edad"
          maxLength={2}
          label="Edad"
          mode={'outlined'}
          error={datos.edadError !== '' && datos.edadError !== null}
          underlineColor="blue"
          value={datos.edad}
          onChangeText={(text) =>
            setDatos({
              ...datos,
              edad: text,
              edadError: validateFields('edad', text),
            })
          }
        />
        <HelperText
          type="error"
          visible={datos.edadError !== '' && datos.edadError !== null}>
          {datos.edadError}
        </HelperText>
```

```
            <TextInput
              placeholder="Ingrese su nombre"
              style={{backgroundColor: 'transparent'}}
              label="Nombre paciente"
              mode={'outlined'}
              error={datos.nombreError !== '' && datos.nombreError !== null}
              underlineColor="blue"
              value={datos.nombre}
              onChangeText={(text) =>
                setDatos({
                  ...datos,
                  nombre: text,
                  nombreError: validateFields('nombre', text),
                })
              }
            />
            <HelperText
              type="error"
              visible={datos.nombreError !== '' && datos.nombreError !== null}>
              {datos.nombreError}
            </HelperText>
          </Card>
        </View>
        <Button
          compact={true}
          color="#5AADF1"
          mode="contained"
          onPress={crearPaciente}>
          <Text>{datos.boton}</Text>
        </Button>
        <Separator />
        <Borrar />
      </View>
    );
  };

const styles = StyleSheet.create({
  separator: {
    marginVertical: 8,
    borderBottomColor: '#737373',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
});

export default NuevoPaciente;
```

ANEXO T: Código fuente del archivo “Grafica.js”

```
import {
  View,
  Text,
  Alert,
  Image,
  StyleSheet,
  Dimensions,
  ScrollView,
  TouchableOpacity,
} from 'react-native';
import React, {useEffect, useState} from 'react';
import {LineChart} from 'react-native-chart-kit';
import {
  Provider as PaperProvider,
  Button,
  ActivityIndicator,
  Colors,
  Badge,
  List,
} from 'react-native-paper';
import {useNavigation, useIsFocused} from '@react-navigation/native';
import moment from 'moment';

const Grafica = ({navigation, route}) => {
  const navigate = useNavigation();
  const windowWidth = Dimensions.get('window').width;
  const [grafica, setGrafica] = useState(true);
  const [prescripciones, setPrescripciones] = useState(false);
  const [datos, setDatos] = useState();
  const [fechaInicio, setInicio] = useState(
    moment(moment(new Date()).subtract(7, 'day'), 'America/Guayaquil'),
  );
  const [fechaFinal, setFinal] = useState(
    moment(new Date(), 'America/Guayaquil'),
  );
  const [noti, setNoti] = useState([]);
  const [cargando, setCargando] = useState(true);
```

```
useEffect(() => {
  getDatosGrafica();
}, []);

const getDatosGrafica = async () => {
  await fetch('http://192.168.1.11:8069/api/grafica', {
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    method: 'POST',
    body: JSON.stringify({
      res_paciente: route.params,
      start_date: new Date(
        moment(fechaInicio).set({h: 0, m: 0}),
      ).toISOString(),
      end_date: new Date(
        moment(fechaFinal).set({h: 23, m: 59}),
      ).toISOString(),
    }),
  })
    .then((response) => response.json())
    .then((responseJson) => {
      setNoti(responseJson.result.notificacion);
      if (
        Array.isArray(responseJson.result.data.labels) &&
        responseJson.result.data.labels.length === 0
      ) {
        setDatos({
          labels: ['0'],
          datasets: [
            {
              id: 1,
              data: [0],
            },
            {
              id: 2,
              data: [0],
            },
          ],
          legend: ['No hay datos para mostrar'], // optional
        });
```

```
      } else {
        setDatos({
          labels: responseJson.result.data.labels,
          datasets: [
            {
              id: responseJson.result.data.datasets[0].id,
              data: responseJson.result.data.datasets[0].data,
              color: (opacity = 1) => `rgba(17, 112, 201, ${opacity})`,
            },
            {
              id: responseJson.result.data.datasets[1].id,
              data: responseJson.result.data.datasets[1].data,
              color: (opacity = 1) => `rgba(201, 17, 17 ,  ${opacity})`,
            },
          ],
          legend: ['SpO2','BPMs'], // optional
        });
      }
      setCargando(false);
    })
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! ☹ Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};

const notiLeido = async () => {
  const values = {
    res_paciente: route.params,
  };
  await fetch('http://192.168.1.11:8069/api/actNoti', {
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
    method: 'PUT',
    body: JSON.stringify(values),
  })
    .then((response) => response.json())
    .then((responseJson) => {})
    .catch((error) => {
      if (error) {
        Alert.alert(
          '¡Error! ☹ Tiempo de espera agotado',
          'No se ha podido realizar la conexion al servidor. Revise su conexion a internet y vuelva a intentarlo.',
          [
            {
              text: 'OK',
            },
          ],
        );
      }
    });
};
```

```
const Renderizado = () => {
  if (grafica) {
    return (
      <View>
        <LineChart
          data={datos}
          width={windowWidth}
          height={250}
          chartConfig={chartConfig}
          verticalLabelRotation={15}
          bezier
          renderDotContent={({x, y, index, id, indexSet}) => (
            <View key={`${id}${index}${indexSet}`}>
              {datos.datasets[indexSet].id === id && (
                <Text
                  key={datos.datasets[indexSet].id}
                  style={{position: 'absolute', top: y, left: x}}>
                  {datos.datasets[indexSet].data[index]}
                </Text>
              )}
            </View>
          )}
        />
      </View>
    );
  } else if (prescripciones) {
    return <Notificaciones />;
  }
};

const Separator = () => <View style={styles.separator} />;
```

```
const MapeoNotis = () => {
  return noti.datos.map((val, index) => {
    return (
      <View key={index}>
        <Button style={{nextFocusDown: true}}>
          <Text style={{fontSize: 20}}>
            {val.fecha} {val.mensaje}
          </Text>
        </Button>
      </View>
    );
  });
};

const Notificaciones = () => {
  return (
    <View style={{height: 280, width: windowWidth}}>
      <ScrollView>
        <List.Section
          style={{
            borderWidth: 0.5,
            backgroundColor: '#F1EFEF',
            borderTopLeftRadius: 10,
            borderTopRightRadius: 10,
            borderBottomLeftRadius: 10,
            borderBottomRightRadius: 10,
          }}>
          <List.Subheader style={{fontSize: 30}}>
            Notificaciones:
          </List.Subheader>
          <MapeoNotis />
        </List.Section>
      </ScrollView>
    </View>
  );
};
```

```
return cargando ? (
  <View style={{alignItems: 'center', justifyContent: 'center', flex: 1}}>
    <ActivityIndicator
      animating={true}
      color={Colors.red800}
      size={'large'}
    />
  </View>
) : (
  <View
    style={{
      backgroundColor: '#fff',
      justifyContent: 'center',
    }}>
    <Image
      style={{resizeMode: 'stretch', width: 1000, height: 120}}
      source={{
        uri: 'https://www.uta.edu.ec/v3.2/uta/images/header.png',
      }}
    />
    <Renderizado />
    <View
      style={{
        backgroundColor: '#fff',
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center',
      }}>
      <Text style={{fontSize: 25}}>Desde: </Text>
      <Button
        color="#C1CBF4"
        compact={true}
        mode="contained"
        onPress={() => {
          if (moment(fechaFinal).isAfter(fechaInicio))
            setInicio(moment(fechaInicio).add(1, 'day').format('YYYY-MM-DD'));
        }}>
        +
      </Button>
      <Text style={{fontSize: 20}}>
        {moment(fechaInicio).format('DD-MM-YYYY')}
      </Text>
```

```
  <Button
    color="#C1CBF4"
    compact={true}
    mode="contained"
    onPress={() =>
      setInicio(
        moment(fechaInicio).subtract(1, 'day').format('YYYY-MM-DD'),
      )
    }>
    -
  </Button>
  <Text style={{fontSize: 25}}> Hasta: </Text>
  <Button
    color="#C1CBF4"
    compact={true}
    mode="contained"
    onPress={() =>
      setFinal(moment(fechaFinal).add(1, 'day').format('YYYY-MM-DD'))
    }>
    +
  </Button>
  <Text style={{fontSize: 20}}>
    {moment(new Date(fechaFinal), 'America/Guayaquil').format(
      'DD-MM-YYYY',
    )}
  </Text>
  <Button
    color="#C1CBF4"
    compact={true}
    mode="contained"
    onPress={() => {
      if (moment(fechaInicio).isBefore(fechaFinal))
        setFinal(
          moment(fechaFinal).subtract(1, 'day').format('YYYY-MM-DD'),
        );
    }}>
    -
  </Button>
</View>
<Separator />
```

```
        <View style={{justifyContent: 'center', alignItems: 'center'}}>
          <Button
            style={{width: 150}}
            color="#C1CBF4"
            compact={true}
            mode="contained"
            onPress={getDatosGrafica}>
            Buscar
          </Button>
        </View>
        <Separator />
        <View style={{flexDirection: 'row', justifyContent: 'space-evenly'}}>
          <Button
            compact={true}
            color="#C1CBF4"
            mode="contained"
            onPress={() => {
              setGrafica(true);
              setPrescripciones(false);
            }}>
            Gráfico
          </Button>
          <View style={{flexDirection: 'row', alignItems: 'center'}}>
            <Button
              compact={true}
              color="#C1CBF4"
              mode="contained"
              onPress={() => {
                setPrescripciones(true);
                setGrafica(false);
                notiLeido();
              }}>
              <Text>Notificaciones</Text>
            </Button>
            <Badge size={35}>{noti.contador}</Badge>
          </View>
        </View>
      </View>
    );
};

const chartConfig = {
  barPercentage: 1,
  backgroundColor: '#fff',
  backgroundGradientFrom: '#fff',
  backgroundGradientTo: '#fff',
  color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
  labelColor: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
  useShadowColorFromDataset: false, // optional
};

const styles = StyleSheet.create({
  separator: {
    marginVertical: 5,
    marginHorizontal: 400,
    borderBottomColor: '#0C3ACA',
    borderBottomWidth: StyleSheet.hairlineWidth,
  },
});

export default Grafica;
```

ANEXO U: Encuesta realizada a profesionales de la salud

Sistema de telemedicina

El objetivo de esta encuesta es el de conocer el beneficio que el sistema de telemedicina de monitorización de BPMs y niveles de SpO2 en un ambiente Smart TV, ofrece al sector de la salud.

* Requerido

1. ¿Qué tan importantes son los datos que el sistema presenta para una valoración médica? *

	1	2	3	4	5	
nada importantes	◯	◯	◯	◯	◯	muy importantes

2. ¿Qué tan útiles son las herramientas que proporciona el sistema? *

	1	2	3	4	5	
nada útiles	◯	◯	◯	◯	◯	muy útiles

3. ¿Qué tan sencilla es la plataforma Web en su manejo? *

	1	2	3	4	5	
nada sencilla	◯	◯	◯	◯	◯	muy sencilla

4. ¿Implementaría este sistema en su lugar de trabajo? *

Si ◯

No ◯

ANEXO V: Precios de las instancias ofrecidas por el servicio de AWS

Tipo de instancia	vCPUs	Arquitectura	Memoria (MiB)	Rendimiento de red	Precio Linux bajo demanda	Precio Windows bajo demanda
t2.nano	1	i386, x86_64	512	Baja a moderada	0.0058 USD por hora	0.0081 USD por hora
t2.micro	1	i386, x86_64	1024	Baja a moderada	0.0116 USD por hora	0.0162 USD por hora
t2.small	1	i386, x86_64	2048	Baja a moderada	0.023 USD por hora	0.032 USD por hora
t2.medium	2	i386, x86_64	4096	Baja a moderada	0.0464 USD por hora	0.0644 USD por hora
t2.large	2	x86_64	8192	Baja a moderada	0.0928 USD por hora	0.1208 USD por hora
t2.xlarge	4	x86_64	16384	Moderada	0.1856 USD por hora	0.2266 USD por hora
t2.2xlarge	8	x86_64	32768	Moderada	0.3712 USD por hora	0.4332 USD por hora