PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Diseño e implementación de una plataforma IoT para el control y monitoreo remoto del banco de psicrometría del Laboratorio de Energía de la PUCP

Tesis para obtener el título profesional de ingeniero de las telecomunicaciones

AUTOR:

Jose Anthony Garcia Macavilca

Asesor:

César Stuardo Lucho Romero

Lima, diciembre, 2022

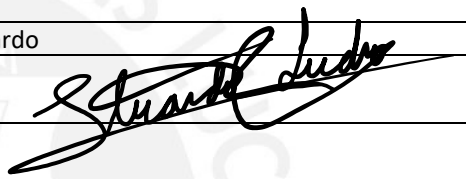
Informe de Similitud

Yo, Cesar Stuardo Lucho Romero, docente de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis titulada Diseño e implementación de una plataforma IoT para el control y monitoreo remoto del banco de psicrometría del Laboratorio de Energía de la PUCP del autor Jose Anthony Garcia Macavilca, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 14%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 05/07/2023.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:5/7/2023

Apellidos y nombres del asesor: Lucho Romero, Cesar Stuardo	
DNI:70326504	Firma
ORCID: 0000-0002-5749-2689	



RESUMEN

La pandemia generada por la Covid 19 generó que las universidades dejen de brindar clases presenciales y que los alumnos dejen de interactuar con los equipos que dichas universidades brindan como complemento de sus aprendizajes. Por otro lado, el internet de las cosas es la tecnología que permite conectar dispositivos a internet y las plataformas IoT las que permiten a un usuario final el monitoreo y control remoto de dichos dispositivos. En ese sentido, el objetivo principal de la tesis es el diseño e implementación de una plataforma IoT que permita el monitoreo y control remoto desde la nube del banco de psicrometría, el cual es un equipo mecánico ubicado dentro del Laboratorio de Energía de la PUCP, a un alumno a través de una interfaz web para la realización de laboratorios remotos. Para ello, se presenta la situación del Laboratorio de Energía en pandemia y se describe al equipo en cuestión. Luego, se presenta los requerimientos a cumplir, se plantea el diseño de la arquitectura y se realiza la implementación de la plataforma. También, se realiza pruebas de rendimiento y se muestra los resultados del uso de la plataforma desarrollada por alumnos en laboratorios reales. Por último, se destacan las conclusiones claves, como el diseño e implementación exitosa de la plataforma IoT. Adicionalmente, se muestra que la plataforma está siendo utilizada por estudiantes de la PUCP a la fecha en la que se escribe la tesis.

Palabras clave

plataforma IoT, control y monitorización remota, laboratorio remoto

DEDICATORIA

A mi madre Enedina, por ser la mujer que me enseño que con esfuerzo y dedicación todo es posible, y ser la que me motiva a esforzarme día a día.

A mi hermano menor Jhosimar, por ser mi alegría.

AGRADECIMIENTOS

Agradezco a mi Asesor Stuardo Lucho, quien me apoyo y motivo desde el primer momento en la que surgió la idea de propuesta de tesis.

Agradezco al gran equipo que conforma el Laboratorio de Energía, que me brindo la mayor de las disponibilidades para realizar las pruebas con el equipo de banco de psicrometría. En especial, agradezco a los ingenieros Fernando Jiménez, Ronald Mas y José Miguel Pérez, quienes me apoyaron logísticamente para realizar dichas pruebas, y a los ingenieros Diego Pichilingue y Arturo Berastain, quienes apoyaron con el despliegue de sensores que facilitó el envío de datos a la plataforma implementada.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El internet de las cosas (IoT) es una tecnología que está tomando relevancia en la actualidad, ya que permite que diferentes dispositivos puedan estar conectados a internet. Asimismo, las plataformas IoT permiten a los usuarios finales poder interactuar con dichos dispositivos y, de esta manera, poder controlarlos y monitorearlos de manera remota.

Por otra parte, los alumnos tuvieron que pasar de controlar los equipos mecánicos, como lo es el banco de psicrometría, dentro de la PUCP a utilizar simuladores para el desarrollo de sus laboratorios debido a la pandemia. No obstante, las plataformas IoT podrían permitir a un alumno controlar y monitorear dichos equipos de manera remota desde cualquier lugar en donde se encuentren conectados a internet.

Por lo tanto, el objetivo principal de esta tesis es diseñar e implementar una plataforma IoT que permita el monitoreo y control remoto desde la nube del banco de psicrometría del Laboratorio de Energía de la PUCP a un alumno a través de una interfaz web para la realización de laboratorios remotos.

El desarrollo de la presente tesis tiene como motivación el poder desarrollar una alternativa de solución al uso de un equipo mecánico en un contexto de pandemia como la del Covid-19 para permitir la continuidad de la enseñanza universitaria de las ingenierías, las cuales requieren necesariamente el desarrollo de laboratorios como complemento de aprendizaje. Asimismo, el reto de lograr desarrollar dicha alternativa poniendo en práctica los conocimientos adquiridos a lo largo de toda la carrera de Ingeniería de las Telecomunicaciones significa una motivación adicional para el autor.

La metodología utilizada en la tesis es la metodología general del diseño de ingeniería aplicada a un problema sentido e importante de las telecomunicaciones y se desarrolla en cinco capítulos.

En el primer capítulo se desarrolla los impactos de la Covid 19 en las universidades del Perú y, en especial, en la PUCP. Asimismo, se presenta al equipo mecánico banco de psicrometría, el cual se encuentra en el Laboratorio de Energía de la PUCP.

En el segundo capítulo se presenta el marco teórico, los criterios para elección de una plataforma u optar por la implementación propia, las plataformas IoT existentes y una comparativa entre las plataformas existentes y la implementación propia. Finalmente, se desarrolla el estudio del arte en donde se presenta las diferentes implementaciones realizadas de las tecnologías IoT para el desarrollo de laboratorios remotos.

En el tercer capítulo se presenta la propuesta de diseño y la arquitectura de la plataforma IoT a implementar basados en una lista los requerimientos brindados por personal del Laboratorio de Energía.

En el cuarto capítulo se muestra la implementación y pruebas de rendimiento realizadas. Asimismo, se presenta los resultados de laboratorios remotos reales realizados con estudiantes que utilizaron la plataforma implementada, las configuraciones finales realizadas para producción y el estado actual de la plataforma.

En el quinto capítulo se realiza el análisis de factibilidad técnica, económica, ambiental, de salud pública y social de la plataforma IoT desarrollada.

Adicionalmente, se presenta las conclusiones, la bibliografía y los anexos de la tesis.

Finalmente, la plataforma IoT diseñada e implementada tiene como logro principal el haber sido utilizada para el desarrollo de laboratorios con el equipo mecánico mencionado durante todo un semestre académico sin presentar inconvenientes, lo que demuestra el éxito del trabajo desarrollado.

Capítulo 1. Situación del uso del banco de psicrometría de la PUCP en el contexto de Covid-19

El presente capítulo se presenta el área de especialización en las telecomunicaciones a la cual está enfocada la tesis, el estado de la enseñanza de las ingenierías en pandemia en la PUCP, se presenta al banco de psicrometría de la PUCP, los requerimientos de la futura plataforma IoT, una comparación entre las plataformas existentes y la implementación propia, los objetivos de la tesis, la motivación personal del autor y la importancia de la solución propuesta para las telecomunicaciones desde el punto de vista técnico, social y educacional.

1.1 Área de especialización en las telecomunicaciones

Las áreas de la carrera de Ingeniería de las telecomunicaciones en las que la presente tesis está enfocada son las áreas de sistemas flexibles y software para telecomunicaciones, e Internet de las cosas (IoT). Los conocimientos adquiridos de la primera área de especialización fueron enseñados en los cursos de Software para telecomunicaciones 1, actualmente llamado Ingeniería web para telecomunicaciones, y Gestión de servicios de TICS. En dichos cursos se

realizaron proyectos de implementación de servicios Tics, lo cual permitieron adquirir conocimientos en el planteamiento y desarrollo de proyectos involucrados con las Tics. Asimismo, los conocimientos adquiridos para la segunda área fueron enseñados en el curso electivo de Temas avanzados en servicios y aplicaciones. Los cursos antes mencionados ayudaron al desarrollo de la competencia de desarrollo e implementación de servicios y aplicaciones de telecomunicaciones y sistemas de distribución de contenidos sobre Internet, web móvil y video bajo demanda e Internet de las Cosas (IoT) que cumplan con criterios de seguridad, confiabilidad y bajo retardo.

1.2 Enseñanza de las ingenierías en pandemia

La aparición de COVID-19 en el Perú representó un gran reto para el sector educativo universitario, ya que debido al aislamiento social obligatorio se necesitó plantear maneras diferentes al tradicional de dictado de clases. En este sentido, la Pontificia Universidad Católica del Perú (PUCP) también tuvo que migrar la modalidad de enseñanza presencial, a la educación a distancia [1]. Asimismo, las clases y prácticas de las especialidades de Ingeniería PUCP pasaron, en pocos meses, de la presencialidad y del manejo de maquinarias al uso de software y de laboratorios virtuales en más del 95% de cursos y prácticas de laboratorio [2]. Uno de los laboratorios en realizar la virtualización de sus experiencias fue el Laboratorio de Energía de la sección de Ingeniería Mecánica de la PUCP, el cual generó sus propios simuladores, tal como se observa en la Figura 1.1. Para esto, primero se desarrolló la recolección de la data generada por sus equipos, ya que se buscó trabajar con data real del equipamiento a ser virtualizado. Segundo, a través de modelos matemáticos y con la data recolectada, se implementó un simulador el cual buscaba generar resultados que un equipo dentro del laboratorio generaría en un laboratorio presencial [3]. Uno de los equipos virtualizados fue el banco de psicrometría.

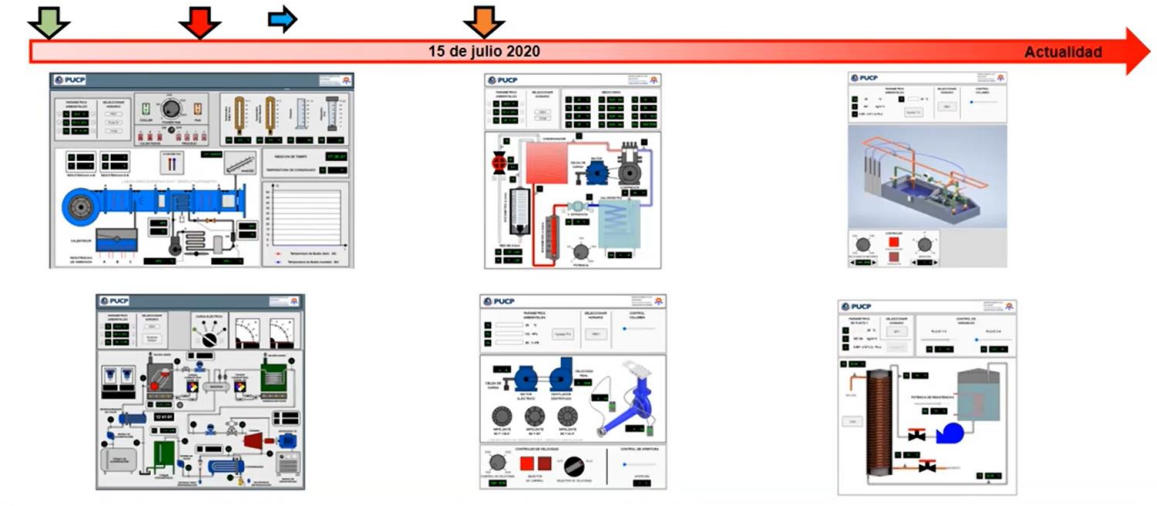


Figura 1.1 - Simuladores creados por el equipo del Laboratorio de energía de la PUCP
Fuente:[3]

1.2.1 Banco de psicrometría del Laboratorio de Energía de la PUCP

El banco de psicrometría es un módulo educativo que simula el proceso de acondicionamiento de aire y permite la compresión y análisis de los procesos psicrométricos por medio de experimentos [4]. La PUCP cuenta con un banco de Psicometría que se encuentra ubicado en el Laboratorio de energía, en el pabellón de U de la sección de Ingeniería Mecánica. Este equipo es utilizado por aproximadamente 200 alumnos en el desarrollo de sus laboratorios, la parte práctica de sus estudios, en los cursos MEC245 (Laboratorio de Termodinámica General), MEC286 (Transferencia de calor) y MEC208 (Termodinámica 2). Sin embargo, debido a las características mecánicas y de dimensión 1.5 m x 2 m x 1 m del equipo los alumnos deben estar presentes para poder interactuar con el mismo. Es por ello, y como se mencionó anteriormente, se realizó su virtualización por medio de un simulador. Así, se permitió que los alumnos que tengan programado sus laboratorios que utilicen este equipo en su desarrollo puedan realizar su laboratorio, pero utilizando el simulador. Sin embargo, según Fernando Jiménez, actual jefe del Laboratorio de Energía, el siguiente reto es dejar de utilizar los simuladores y que los alumnos sean capaces de controlar los equipos, entre ellos el banco de psicrometría, de forma remota, desde sus casas o desde cualquier lugar que se encuentren conectados a internet [3].

Figura 1.2 - Banco de psicrometría de la PUCP
Fuente: Fotografía propia

Sin embargo, ¿cómo lograr que un alumno pueda controlar el banco de psicrometría remotamente?

Para responder a la última pregunta planteada es necesario notar dos puntos. En primer lugar, se necesita que el equipo esté conectado a internet. Cuando un dispositivo, equipo, o una "cosa" está conectado a internet, se está hablando de internet de las cosas o IoT. El internet de las cosas describe la red de objetos físicos, las "cosas", que están embebidos con sensores, software y otras tecnologías con el propósito de conectarse e intercambiar *data* con otros dispositivos y sistemas a través de internet [5]. En segundo lugar, una vez el banco de psicrometría esté conectado a internet, se necesita que el alumno pueda monitorear y controlar el equipo remotamente. Esto puede lograrse a través de una plataforma IoT.

1.3 Plataformas IoT

Una plataforma IoT puede tener una diferente definición según la perspectiva: la perspectiva de la investigación científica, la perspectiva de mercado y la perspectiva de código abierto. Sin embargo, se puede llegar a una definición considerando las tres perspectivas. Una plataforma

IoT es la interfaz entre los dispositivos IoT y los usuarios finales que provee la administración de dispositivos, conexiones, servicios y de *data*, usualmente a través de la nube. La plataforma ofrece almacenamiento, gestión, análisis y visualización de datos, gestión de seguridad de datos, tanto en usuarios como en los dispositivos, y capacidad de integración. Asimismo, ofrece algunas herramientas, como APIs y servicios, para la creación, implementación y desarrollo de aplicaciones para dispositivos IoT y usuarios finales [6]. Además, debido a que la plataforma IoT se encuentra entre los dispositivos IoT y los usuarios finales usualmente también se le hace referencia a ellas como el *middleware* y la infraestructura que permite a los usuarios finales interactuar con los objetos inteligentes [7]. Por este motivo, se propone el uso de una plataforma IoT.

1.4 Requerimientos de la solución

Una vez propuesto el uso de una plataforma IoT para el control remoto del banco de psicrometría es necesario el planteo de requerimientos a ser cumplidos por la futura plataforma a ser utilizada. Basados en el estudio de campo realizado en [8] y la solicitud de una lista de requerimientos solicitado al equipo dentro del Laboratorio de Energía se plantea los siguientes requerimientos (se detallarán a detalle en capítulos posteriores):

- Gestión de usuarios basado en tres roles: alumnos, jefe de práctica y administrador.
- Automatización del acceso a la plataforma por los alumnos basado en fecha y horas.
- Asignación por parte de los jefes de práctica de los interruptores que pueden ser controlados por un alumno en tiempo real.
- Visualización del comportamiento de la data generada por el equipo en tiempo real.
- Visualización de medidores que no generen *data* (como un nanómetro) mediante cámaras.

1.5 Objetivos de la tesis

1.5.1 Objetivo general

La presente tesis tiene como objetivo principal el diseño e implementación de una plataforma IoT que permita el monitoreo y control remoto desde la nube del banco de psicrometría del Laboratorio de Energía de la PUCP a un alumno a través de una interfaz web para la realización de laboratorios remotos.

1.5.2 Objetivos específicos

Además del objetivo general, se plantea los siguientes objetivos específicos:

- La plataforma IoT debe permitir la gestión de usuarios basado en tres roles: alumno, jefe de práctica y administrador.
- La plataforma IoT debe permitir la automatización del acceso a la plataforma por los alumnos basado en fechas y horas.
- La plataforma IoT debe permitir la asignación en tiempo real de los interruptores que pueden ser controlados por un alumno.
- La plataforma debe permitir la visualización del comportamiento de la data generada por el equipo en tiempo real.
- La plataforma debe permitir la visualización a través de cámaras de aquellos medidores que no puedan generar *data*.

1.6 Motivación personal

La presente tesis se desarrolla como respuesta al reto de controlar remotamente el equipo del banco de psicrometría de la PUCP por los alumnos para el desarrollo de sus laboratorios como una alternativa diferente al uso de simuladores en un contexto de pandemia como la del Covid-

19. Desde el punto de vista del autor, el ofrecer al alumno la oportunidad de controlar remotamente el equipo antes mencionado de manera remota es una mejor alternativa que el uso de simuladores, pues el alumno realiza sus experiencias basadas en acciones en tiempo real y no con un simulador que tiene configuraciones previamente definidas. Asimismo, la plataforma podrá ser utilizada como herramienta ante una posible nueva ola de contagios que eviten el retorno de los alumnos a la universidad. Por ejemplo, a fecha de ser escrita la presente tesis, el ministro de salud del Perú ha anunciado el inicio de la cuarta ola del Covid-19 [9]. Finalmente, el autor considera que el diseño e implementación de la plataforma IoT podrá ser utilizado por alumnos de distintas universidades, fuera de Lima, que no cuenten con equipos similares al banco de psicrometría para el desarrollo de experiencias que contribuyan a sus aprendizajes. De esta forma, se contribuye con la reducción de la brecha educativa existente en nuestro país.

1.7 Importancia del problema para las telecomunicaciones

1.7.1 Importancia técnica

La educación a distancia es una realidad y la pandemia sólo ha ayudado a apresurar la búsqueda de soluciones novedosas para mejorar el desarrollo de una educación no presencial de calidad. El diseño e implementación de la plataforma IoT propuesta en la presente tesis puede contribuir como experiencia y precedente para el desarrollo de futuros proyectos tecnológicos en los laboratorios de la PUCP y otras universidades que permitan que un alumno interactúe remotamente con los equipos sin necesidad de estar presente dentro de un laboratorio.

1.7.2 Importancia social

Desde el punto de vista social, el tener una herramienta como la propuesta en la presente tesis puede ayudar no solo a los alumnos pertenecientes a la PUCP, sino a alumnos de distintas universidades que no cuenten con un equipo de alto costo como el banco de psicrometría para

realizar experiencias de laboratorio. Así, se puede crear un esquema de cooperación entre universidades que permitan el acceso a sus recursos a un gran número de estudiantes pertenecientes a las universidades de dicha cooperación [10]. Por ejemplo, la PUCP pertenece a la Red Peruana de Universidades la cual tiene como objetivo promover la cooperación interinstitucional entre las universidades miembros, con la finalidad de que los miembros de las comunidades universitarias desarrollen, de manera conjunta, proyectos y actividades para el desarrollo regional y nacional [11]. Así, se puede invitar a los alumnos de dichas universidades a utilizar la plataforma IoT desde el lugar en donde se encuentren conectados a internet. De otro modo, tendrían que viajar a Lima e ingresar a la PUCP para poder realizar las mismas experiencias, lo cual significa un gasto económico para ellos y, además, un peligro debido al contexto de pandemia aún vigente en el país. En suma, se ayuda en la reducción de la brecha educativa vigente dentro del país, que, en la mayoría de casos, es por temas económicos.

1.7.3 Importancia educacional

La implementación de la plataforma IoT de la presente tesis puede permitir una transformación de la enseñanza universitaria dentro de la PUCP, pues brindará una herramienta que puede ser utilizada en el desarrollo de una clase teórica. Por ejemplo, un profesor sería capaz de controlar un equipo remotamente desde un salón de clases y visualizar la data generada para su evaluación en tiempo real por todos los alumnos de la clase, de manera conjunta con el profesor, sin necesidad de que se movilicen hacia el lugar físico donde se encuentre dicho equipo, ya que el control total podría realizarse desde la nube a través de internet y con una simple laptop.

Capítulo 2. Marco teórico y estado del arte

2.1 Marco teórico

2.1.1 Arquitectura de una plataforma IoT

La arquitectura de una plataforma IoT puede representarse por 3 capas principales [12] [13] [14]:

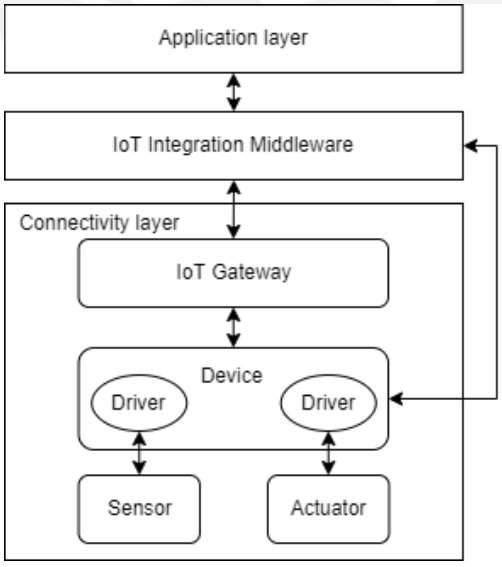


Figura 2.1 Arquitectura de una plataforma IoT
Fuente: [12] [13] *y* [14]

a. Capa de conectividad

La capa de conectividad está conformada por el conjunto de dispositivos que adquieren data a través de sensores y que se comunican entre ellos a través de diferentes protocolos como por ejemplo Wifi, Bluetooth, Zigbee, Z-wave, LoRaWAN, etc [13]. Esta capa está conformada por los siguientes componentes:

- **Sensor**: Componente de *hardware* usado para medir parámetros del entorno físico y convertirlos en señales eléctricas [14]. Un ejemplo común son los sensores que miden temperatura y humedad de una habitación.
- **Actuador**: Componente de *hardware* que controla o manipula el entorno físico, pues convierte las señales eléctricas en una acción física [14].
- **Dispositivo**: Componente de *hardware* al que los sensores y actuadores están conectados alámbricamente o inalámbricamente; sin embargo, los sensores y actuadores también pueden estar integrados en estos dispositivos. El "dispositivo" es el encargado de recibir la *data* de los sensores y, también, de enviar comandos hacia los actuadores. Ejemplos comunes son las placas Arduino, Raspberry Pis, Banana Pis o BeagleBones [14].
- **IoT Gateway**: Componente que permite a un dispositivo que utiliza protocolos de comunicación que no soporten el protocolo IP (como por ejemplo Zigbee o Z-wave) pueda comunicarse con la siguiente capa, el *IoT middleware* [13]. El *IoT gateway* provee las funcionalidades y tecnología requeridas para "traducir" diferentes protocolos y reenviar la comunicación entre los dispositivos y otros sistemas [14]. Por ejemplo, cuando se tiene sensores dentro de una red LoRaWan (protocolo de red que usa la tecnología LoRa) pero la plataforma IoT es un Endpoint MQTT, entonces se necesitaría un gateway.

b. IoT Middleware

La capa *IoT Middleware*, también conocida como el *software back-end*, es el componente central de una plataforma IoT [13]. Esta capa tiene múltiples funcionalidades, siendo la principal recibir la *data* enviada por los dispositivos conectados para procesarla y enviarla a las aplicaciones conectadas a esta capa. De igual manera, puede procesar dicha *data* y, a través de la evaluación de reglas condición-acción, controlar dispositivos en forma de comandos a ser ejecutados por sus respectivos actuadores [14]. Sin embargo, también puede cumplir otras funcionalidades como la de soporte de *firmware updagre support* remoto a los dispositivos, [13], gestión de dispositivos y usuarios, agregación de la *data* recibida, tener un *rule engine* o generar *dashboards*, según sea requerido por la solución IoT [14], aunque, usualmente, los *dashboards* se encuentran en la capa de aplicación. Asimismo, un dispositivo de la capa de conectividad puede conectarse a esta capa directamente, sin la necesidad de un IoT *Gateway*, si cumple las siguientes tres condiciones [14]:

- soporta una apropiada tecnología de comunicación (como el WiFi)
- utiliza un protocolo de transporte apropiado a la tecnología de comunicación utilizada (como HTTP o MQTT)
- usa un formato compatible de *payload* (como el JSON o XML)

Finalmente, esta capa, típicamente, puede accederse utilizando APIs; por ejemplo, las REST API basado en HTTP [14].

c. Capa de aplicación

La capa de aplicación es la que permite al usuario final interactuar con la plataforma IoT [13]. Una “aplicación” es el software que obtiene información y modifica el entorno físico a través de la solicitud de *data* de los sensores o controlar acciones físicas a través de los actuares [14]. Así, se permite visualización de *data*, control de dispositivos, análisis de data, etc [12][13]. Sin embargo, una “aplicación” también puede ser otro IoT *Middleware*, por ejemplo, para la integración de múltiples sistemas [14]. Los servidores web presentan una gran relevancia en

esta capa, ya que la gran mayoría de plataformas IoT tienen algún tipo de interfaz gráfica web, para que el usuario final interactúe con él, que permita el control remoto de los dispositivos conectados al ambiente IoT a través de internet [12].

2.1.2 Diagrama de bloques de una plataforma IoT

Una plataforma IoT está compuesta por un conjunto de funcionalidades. Sin embargo, se puede considerar que son 8 los componentes o bloques que una plataforma IoT debe tener para funcionar eficientemente [15], los cuales puedes observarse en la Figura 2.2.

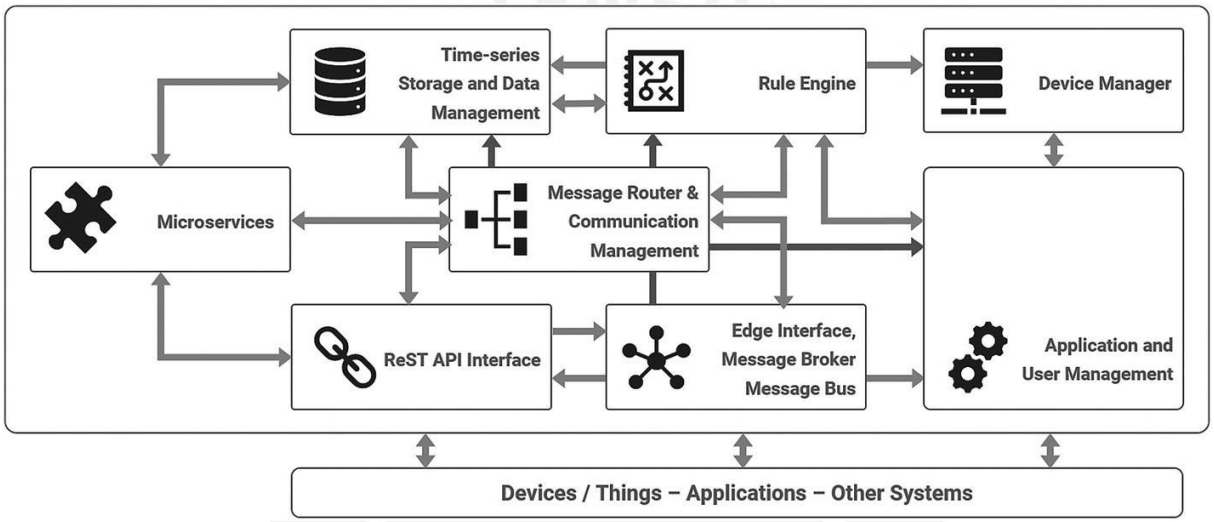


Figura 2.2 Diagrama de bloques de una plataforma IoT
Fuente: [15]

a. Edge Interface, Message Broker y Message Bus

Este módulo es el encargado de interactuar con los dispositivos de la capa de conectividad, de recibir todos sus mensajes y unificarlos para poder enviarlos a un *bus* común de mensajes [15]. Un ejemplo común es un bróker MQTT.

b. Message Router and Communication Management

Una vez que el mensaje se encuentra disponible en el bus de mensajes, el mensaje podría necesitar de atributos extras para ser de utilidad a otros módulos o para ser utilizado por las diferentes aplicaciones conectados a la plataforma. El encargado de realizar dichas

modificaciones a los mensajes, reenviarlos al *bus* de comunicaciones, publicar información contextual extra y otros mensajes después de que el mensaje original ingrese es el administrador de comunicaciones. Asimismo, puede realizar trabajos de conversión de formatos; por ejemplo, conversión de CSV a JSON o de binario a formato texto [15].

c. Time-Series Storage and Data Management

Este bloque es el encargado de almacenar la *data* que está disponible en el *bus* de mensajes de manera secuencial (usualmente en forma de series de tiempo) [15].

d. Rule Engine

El motor de reglas es el encargado de monitorear el *bus* de mensajes y tomar acciones basado en reglas [15]. Por ejemplo, se puede enviar una alerta a un usuario si el sensor de movimiento de su casa detecta movimiento.

e. The REST API Interface

Las Restful APIs son útiles para soportar funciones que no necesiten una conectividad y acceso constante o en tiempo real a la plataforma. Usualmente, las aplicaciones acceden a este módulo; sin embargo, un dispositivo también puede acceder a este módulo cuando sea necesario. Asimismo, se puede implementar una autenticación basado en roles para poder acceder a las APIs [15].

f. Microservices

El módulo de microservicios está conformado por servicios auxiliares que permiten un funcionamiento efectivo de la plataforma. Se puede encontrar ejemplos como mensajería de texto o notificaciones de correo electrónico, verificaciones, *captchas*, autenticación por redes sociales o servicios de pago [15].

g. Device Manager

A medida que la cantidad de dispositivos conectados a una plataforma aumenta la complejidad de mantener un control sobre dichos dispositivos también lo hace, es entonces cuando el módulo de administración de dispositivos puede ayudar. Este módulo provee funcionalidades para administrar los dispositivos como entidades para poder, por ejemplo, listarlos, controlar sus estados actuales, niveles de carga de baterías, detalles del dispositivo, información de su sesión, etc. Asimismo, este módulo también ayuda a realizar actualizaciones *over-the-air* a los dispositivos conectados [15].

h. Application and User Management

El bloque de administración de usuarios y aplicaciones provee funcionalidades similares al del módulo de administración de dispositivos, pero orientado a los usuarios y aplicaciones. Funcionalidades típicas en la administración de usuarios son funciones de administración de contraseñas y credenciales, llaves de accesos, autenticación y autorización. Asimismo, funcionalidades típicas en la administración de aplicaciones son llaves API, credenciales y accesos [15].

2.1.3 Protocolos IoT

Una de las principales características de un entorno IoT es que se encuentran conectados una gran cantidad de dispositivos que constantemente están generando *data*, lo que resulta en un envió enorme de *data* entre dispositivos [16]. Existe diferentes protocolos de comunicación estandarizados; sin embargo, se presentará los dos protocolos más utilizados: CoAP y MQTT [17].

a. Protocolo CoAP

El protocolo CoAP es un protocolo de transferencia web especializado para su uso en nodos y redes con “limitaciones”, como puede ser alta cantidad de pérdidas, ancho de banda limitado o

niveles de potencia de transmisión bajas, como es común en un entorno IoT. Asimismo, CoAP provee un modelo de interacción de *request*/*response* entre *endpoints* [18]. Es decir, está basado en un modelo cliente/servidor similar a HTTP donde el cliente realiza una solicitud al servidor y este le envía una respuesta. De igual manera que en HTTP, los clientes pueden realizar solicitudes utilizando métodos *GET*, *POST*, *PUT* y *DELETE*.

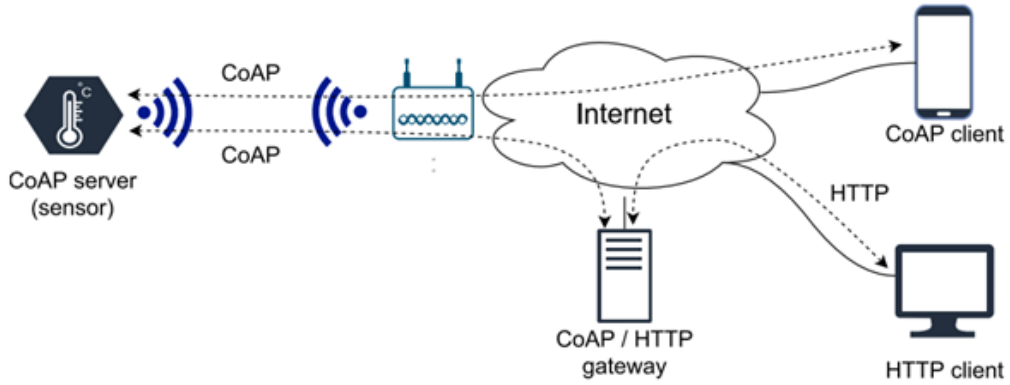


Figura 2.3 Funcionamiento general de un sistema con CoAP
Fuente: [17]

Con respecto a la confiabilidad del envío de mensajes, los *requets* o *responses* pueden estar marcados como "*confirmable*" o "*nonconfirmable*". Los primeros necesitan de un mensaje de *acknowledgement* del receptor para confirmar la recepción del mensaje, mientras que los segundos no necesitan de una respuesta por parte del receptor [19].

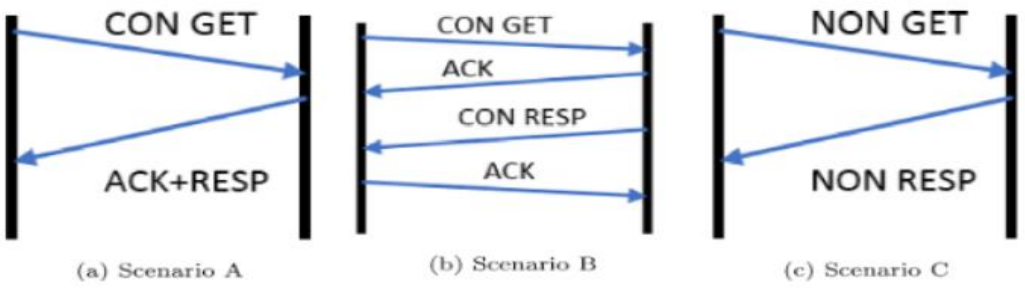


Figura 2.4. Posibles escenarios de un mensaje GET CoAP
Fuente: [17]

El protocolo CoAP está basado en el protocolo UDP (*User datagram protocol*) lo que le permite lograr el envío de información en situaciones, como en una red con mucha congestión, en donde otros protocolos basados en TCP fallarían en completar *handshake* [20].

b. Protocolo MQTT

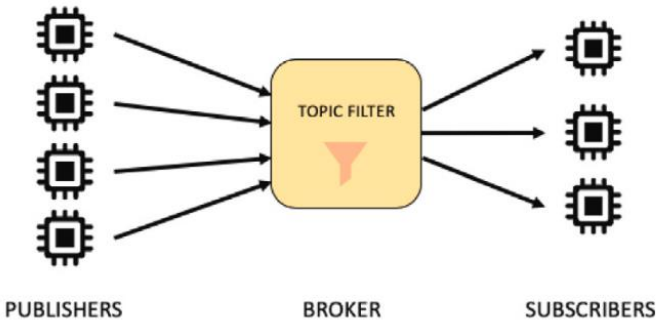


Figura 2.5. Arquitectura de intercambio de mensajes MQTT
Fuente:[17]

MQTT es un protocolo de transporte ligero, abierto, simple y diseñado para una implementación sencilla basado en una arquitectura de suscripción y publicación (*publish*/*subscribe*). Debido a estas características, su uso en comunicaciones *Machine-to-Machine* (M2M) e Internet de las cosas es ideal, ya que el uso del ancho de banda requerido es importante [21]. En un entorno donde el protocolo MQTT es utilizado, existe un servidor centralizado llamado *bróker*, al cual todos los sensores deben estar conectados. De igual manera, todos los mensajes enviados y/o recibidos se encuentran "etiquetados" con cierto *topic*. A la acción de enviar un mensaje se le llama "publicación" del mensaje y para que un dispositivo pueda recibir cierto mensaje este debe estar "suscrito" al *topic* con el cual el mensaje a ser recibido se encuentra etiquetado, de ahí a que se conoce como una arquitectura basada en "suscripción y publicación". Adicionalmente, cada cliente puede suscribirse a múltiples *topics* y todos los clientes que se suscriben a cierto *topic* reciben un mensaje cuando este es publicado con ese *topic* determinado [19].

En la Figura 2.6 se puede observar un ejemplo con el cual se puede explicar este mecanismo de mensajería. Imaginemos que se tiene un sensor de temperatura en una habitación, un celular y un servidor *back-end*, que puede ser un servidor web, conectados a un broker MQTT. El celular y el servidor se encuentran "suscritos" al *topic* "temperatura". Por su lado, el sensor "publica" una medición de temperatura de 24 °C de la habitación con el *topic* "temperatura" hacia un

broker MQTT. Cuando el *broker* recibe el mensaje publicado, este reenvía dicho mensaje al celular y al servidor web, ya que estos se encuentran suscritos al *topic* “temperatura”.

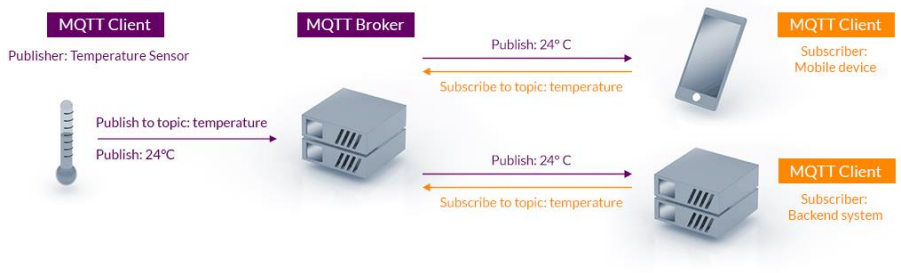


Figura 2.6. Ejemplo de publicación y suscripción
Fuente: [22]

El protocolo MQTT ofrece 3 niveles de QoS: QoS 0, QoS 1 y QoS 2 [21].

- **QoS 0 (a lo mucho se entrega uno)**

El mensaje es enviado de acuerdo a las capacidades de la red. No se realizan retransmisiones o se recibe un mensaje de confirmación del receptor. Así, el mensaje enviado es entregado a lo mucho una vez [17].

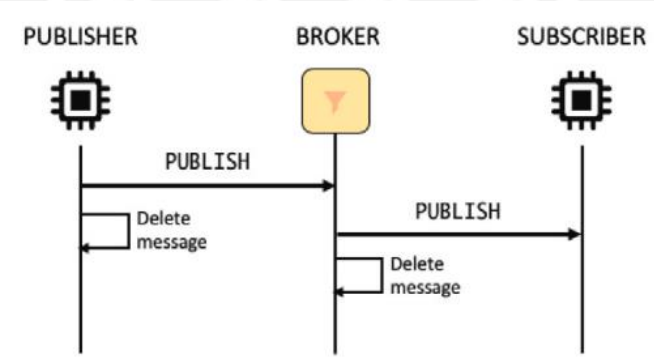


Figura 2.7. QoS de nivel 0
Fuente:[17]

- **QoS 1 (al menos se entrega uno)**

El receptor debe responder como un mensaje de confirmación. Caso contrario, el mensaje es retransmitido. El mensaje puede enviarse varias veces, debido a las retransmisiones, pero se entrega al menos una vez [17].

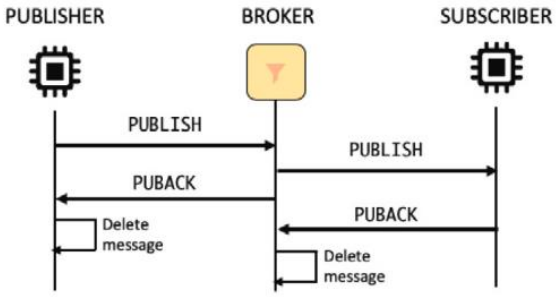


Figura 2.8. QoS de nivel 1
Fuente: [17]

- **QoS 2 (exactamente uno entregado)**

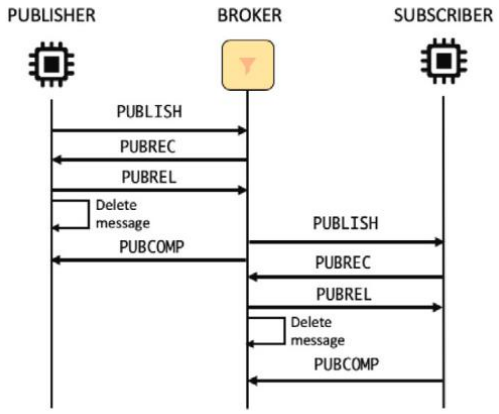


Figura 2.9. QoS de nivel 2
Fuente: [17]

Se realiza un *four-way handshake* que asegura que el mensaje es entregado exactamente una vez [17].

Notamos que a diferencia del protocolo CoAP, MQTT tiene más niveles de QoS, lo que le permite tener transmisiones más confiables. Sin embargo, altos niveles de QoS generan latencia [20], debido al *handshake* para QoS 2, por ejemplo.

2.1.4 Aplicaciones web en tiempo real

Como se mencionó anteriormente, usualmente la capa de aplicación de una plataforma IoT se accede a través de una interfaz web y como las plataformas se caracterizan por brindar data en tiempo real, es necesario un análisis de la evolución de las aplicaciones web en tiempo real.

a. El modelo cliente-servidor tradicional

El modelo cliente-servidor ha sido la columna vertebral de la arquitectura de la web en donde el cliente solicita un recurso (a través de un HTTP *request*) y este es brindado por un servidor remoto (a través de un HTTP *response*) [23]. Sin embargo, se puede notar que para que un cliente obtenga algún recurso, este tiene que necesariamente solicitarlo. Es decir, la comunicación es unidireccional. No obstante, a medida que las aplicaciones web se vuelven más dinámicas e interactivas, los servidores usualmente necesitan enviar recursos al cliente independientemente si este lo solicita [23].

Así, fue necesario la invención de mecanismos para conseguir una comunicación bidireccional entre el servidor y el cliente. De esta manera, surgieron los mecanismos de HTTP *polling* y HTTP *long polling*. Asimismo, el protocolo WebSocket surgió como solución a los inconvenientes que fueron apareciendo a través del tiempo [23].

b. HTTP polling y HTTP long polling

HTTP *polling* consiste en una secuencia de mensajes *request-response* en donde el cliente envía un *request* al servidor y el servidor responde con un nuevo mensaje, si existe alguno, o un mensaje vacío, si no existe ninguno. Después de un tiempo, conocido como *polling interval*, el cliente vuelve a realizar un *request* para descubrir si un nuevo mensaje se encuentra disponible, el proceso se repite constantemente cada *polling Interval* [24]. Existe dos problemas con este método. En primer lugar, cada *request* y *response* tiene cabeceras HTTP, lo que genera un desperdicio del tráfico de red y, en segundo lugar, el envio de nuevos mensajes del servidor al cliente solo se realiza a una frecuencia determinada, cada *polling interval*; es decir, el servidor no está enviando mensajes de manera directa al cliente [23], lo que genera que exista cierto *delay* entre el momento en que un mensaje está disponible y el momento en que este realmente llega al cliente.

Una optimización de la técnica *polling* fue el *long polling*. HTTP *long polling* utiliza la habilidad del servidor de mantener la solicitud del cliente abierta hasta que un nuevo mensaje se encuentre disponible [23]. Así, el servidor no envía respuestas vacías cuando no tiene nuevos mensajes, lo que reduce considerablemente el número de *requests* del cliente cuando nuevos mensajes no se encuentran disponibles [24].

c. WebSockets

El protocolo WebSocket presentó un gran optimización en la cantidad de mensajes HTTP, ya que solo son necesarios un HTTP *request* y un HTTP *response* para poder completarse [23]. Luego, se crea un canal de comunicación bidireccional por la cual tanto el cliente como el servidor pueden enviar mensajes independientemente de que alguno de ellos lo solicite. Los WebSocket *frames* enviados sobre una conexión existente contienen una cabecera, pero dicha cabecera es mucho más pequeña que una cabecera HTTP, ya que usualmente es menor a 8 bytes [23]. Así, WebSockets ha permitido el desarrollo de aplicaciones Web en tiempo real escalables [24].

Se puede hacer uso del protocolo WebSocket por medio de diferentes lenguajes de programación, cada uno con sus propias librerías para poder implementarlo. En [25] se realiza una comparación de diferentes lenguajes de programación y sus librerías para obtener la de mejor *performace* para la implementación de un servidor websocket. Las pruebas realizadas se basan en aplicaciones *echo server* y el parámetro a medir es el *round trip time* total. Los lenguajes de programación puestos a prueba fueron C, C++, C#, Go, Java, Node.js, PHP, Python y Rust. Los resultados demuestran que Node.js demuestra mejor *performace* cuando se trata de implementar un servidor WebSocket. En la Figura 2.10 se muestra que Node.js es la que menos tiempo toma en responder todos los *requests* dada una cantidad de conexiones, siendo 10000 la máxima cantidad de conexiones probada.

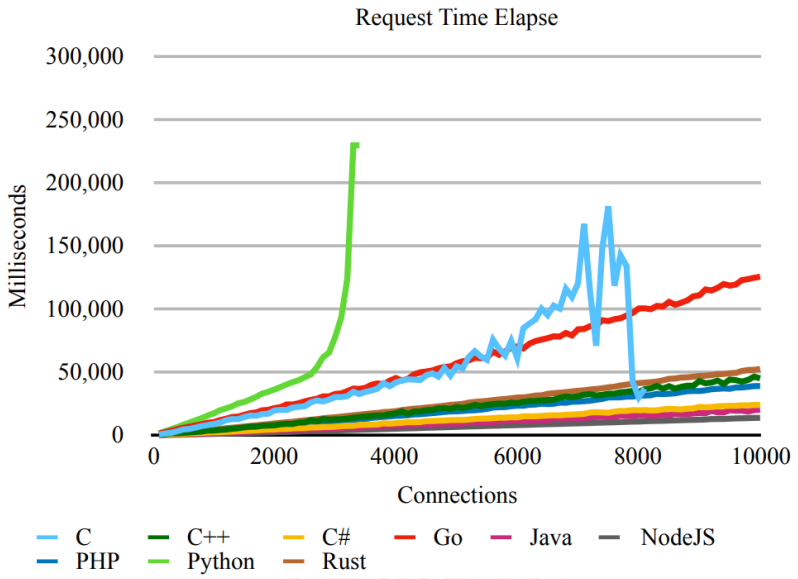


Figura 2.10 Tiempo que le toma al servidor websocket en responder todos los request dada una cantidad de conexiones
Fuente: [25]

Asimismo, se obtuvieron los siguientes hallazgos:

- Fue una de las de menor tiempo mostró en establecer conexión con los 10000 clientes, solo por detrás de C# y Go.
- Fue la que menor tiempo tomó en responder todos los *requests* realizados (5.5 millones de *requets*, en total).
- Fue la que menor tiempo tomo en terminar todo el *test* realizado por el autor.
- Fue la que menor tiempo deja a un cliente en responder.

El motivo de que Node.js demostrará mejor *performance* en las pruebas realizadas fue su naturaleza asíncrona, la cual le permite realizar tareas mientras otras aún no finalizan de realizarse [25].

2.1.5 Gestión de datos en IoT

El uso de aplicaciones IoT incrementa la generación de *data* en tiempo real, lo cual convierte al almacenamiento y administración de *data* en todo un desafío [26].

Una base de datos puede ser definido como una colección estructurada de *data* y el sistema encargado de administrar dicha *data*, los problemas relacionadas a estas o cualquier otro aspecto de una base de datos, es el Sistema de Administración de Bases de datos (DBMS, por sus siglas en inglés) [26]. En el mercado actual, predominan dos tipos principales de bases de datos: las RDMS (DBMS relacionales o bases de datos SQL) y las NoSQL [27].

a. Relational DBMS (RDMS)

Los DBMS relacionales utilizan el leguaje SQL y son los sistemas de base de datos tradicionales [26]. Los RDBMS almacenan *data* en estructuras de tablas las cuales se encuentran relacionadas entre ellas [27], de ahí que se les conoce como "relacionales". Las operaciones realizadas en un RDMS cumplen con las propiedades conocidas como ACID (por su acrónimo del inglés) [28]:

- **Atomicidad**: Todos los cambios de *data* son realizados como si dichos cambios fueran una sola operación. Es decir, todos los cambios se realizan o ninguno lo hace.
- **Consistencia**: La base de datos debe seguir reglas que validen y prevengan corrupciones en cada paso de algún cambio.
- **Aislamiento**: El estado intermedio de un intercambio no es visible a otro intercambio, y como resultado, los intercambios que se ejecutan en concurrencias aparentan estar serializados.
- **Durabilidad**: Los cambios de *data* son persistentes una vez finalizado un intercambio y no son reversibles, aunque se produzca fallos en el sistema.

Algunas de las RDMS más conocidas son MySQL, PostgreSQL, SQL Server, Oracle, Db2 y Informix [27].

b. Base de datos NoSQL

El término NoSQL significa "*Not just SQL*". A diferencia de las bases de datos relacionales, donde la *data* se almacenaba en tablas de filas y columnas, las NoSQL permiten almacenar *data* en diferentes estructuras dinámicas de datos. En general, las bases de datos NoSQL, según la estructura con la que almacenas *data*, se pueden clasificar en 4 tipos [28]:

- **Orientado a columnas**: La *data* se almacena en celdas agrupadas en un numero virtualmente ilimitado de columnas en lugar de filas.
- **Llave-valor**: Utiliza un modelo de datos basado en *arrays* asociativos, conocidos como diccionarios o *maps*. La *data* es representada como una colección de pares llave-valor.
- **Almacenamiento de documentos:** Utiliza documentos para mantener y codificar la *data* en formatos estándares, como XML, YAML, JSON y BSON.
- **Gráficos**: Se representa la *data* en un gráfico que muestra qué tan diferente es un conjunto de datos respecto a otro conjunto.

Una característica de las bases de datos NoSQL es que escalan de mejor manera horizontalmente; es decir, se van agregando servidores a medida que la carga se incrementa. A diferencia de las bases de datos relacionales, en donde se tenía la propiedad ACID, las NoSQL se basan en la teoría CAP, la cual consiste en que un sistema de *data* distribuido garantiza al mismo tiempo solo dos de las siguientes tres propiedades [28]:

- **Consistencia**: Cada solicitud recibe o bien el resultado más reciente o un error.
- **Disponibilidad**: Todas las solicitudes obtienen una respuesta, sin excepción.
- **Tolerancia a partición**: Cualquier retraso o pérdida entre nodos no interrumpe la operación de todo el sistema.

Alguno de los ejemplos de bases de datos NoSQL son MongoDB, Redis, JanusGraph, Etdc, RabbitMQ y Elasticsearch [27].

2.1.6 Apache JMeter

Apache JMeter es una aplicación de *software* de código libre desarrollado en java diseñado para realizar pruebas de carga y medición de rendimiento de aplicaciones Web. Puede ser utilizado para simular grandes cargar a un servidor, grupo de servidores o redes para evaluar el rendimiento bajo diferentes tipos de cargas [29]. Las simulaciones realizadas se basan en un "test plan". El "test plan" describe una serie de pasos a ser ejecutados por JMeter cuando este es ejecutado y está conformado por un grupo de elementos, entre los que se encuentra los "Thread Group", los "HTTP(S) Test Script Recorder" y los "Recording Controller".

Los "Thread Group" son el punto de inicio de todo "test plan" y, como su nombre lo indica, controla el número de hilos que JMeter usará para ejecutar sus pruebas. Este elemento permite configurar la cantidad de hilos a utilizar, el intervalo de inicio de cada hilo y la cantidad de tiempo que se ejecuta una prueba.

Los "HTTP(S) Test Script Recorder" permiten que JMeter pueda interceptar y grabar las acciones realizadas mientras se navega en un navegador web. Este elemento es implementado como un *HTTP(S) proxy server* por lo que es necesario configurar el navegador web para que utilice el *proxy* y capture todas las solitudes HTTPS.

Los "Recording Controller" es el elemento en donde se almacenarán las solicitudes HTTPS capturadas por el servidor *proxy*.

2.2 Criterios para elección de una plataforma u optar por la implementación propia

Una plataforma IoT tiene funcionalidades y características que pueden ser opcionales u obligatorias a ser cumplidas, distinguir entre ambas es indispensable para tomar la decisión de utilizar una plataforma propia u optar por una ya existente, sea de código libre o propietaria. A

continuación, se listan las funcionalidades y capacidades que una buena plataforma IoT debe tener para formar parte del internet de las cosas [15].

- **Costo**: El presupuesto a invertir en una plataforma es esencial, ya que dependiendo de las capacidades requeridas de una plataforma puede elevar el costo de esta, pero en ocasiones el dinero invertido no justifica las funcionalidades y capacidades que esta brinda. Así, una plataforma debe ser lo suficientemente buena para justificar su costo.
- **Escalabilidad**: El concepto de escalabilidad connota la capacidad de un sistema para controlar un número cada vez mayor de elementos u objetos, procesar volúmenes crecientes de trabajo sin dificultad y/o tener potencial de ser ampliado para soportar dicho incremento de volumen de trabajo [30]. Una plataforma debe ser razonablemente fácil de escalar sin interrumpir en la producción existente que se esté realizando.
- **Confiabilidad**: Debido a que la plataforma IoT se encuentra en el *core* de una solución IoT necesita contar con un nivel de confiabilidad lo suficientemente buena dependiendo del tipo de aplicación en donde se utilice la plataforma.
- **Personalización**: La plataforma debe tener un nivel de personalización acorde al contexto que se le aplique, ya que, si esto no se cumple, la empresa debe modificar el producto o servicio para adaptarse a la plataforma, lo cual es esencialmente trabajo en sentido inverso que se debe evitar.
- **Protocolos soportados e interfaces**: Por definición la plataforma IoT se encuentra entre los dispositivos IoT y las aplicaciones que permiten a los usuarios interactuar con dichos dispositivos. Es decir, está entre dos sistemas heterogéneos: dispositivos físicos y *software* en la nube. La plataforma debe tener la capacidad de coordinar con ambos sistemas, orquestarlos y soportar todos los protocolos de comunicación que intervienen en dicha comunicación.

- **Agnóstico del *hardware***: Debido a que el internet de las cosas es esencialmente un grupo heterogéneo de "cosas" conectadas (dispositivos físicos, sistemas de computadoras y *software*), la plataforma deber ser independiente del tipo de dispositivo que pueda conectarse a ella y debe soportar la comunicación de este con la plataforma.
- **Agnóstico de la nube**: Actualmente existen varios proveedores *cloud*, como Amazon Web Services (AWS), Microsoft, Google, etc, pero la plataforma debe poder ser desplegada en cualquier proveedor sin afectar el funcionamiento de este.
- **Arquitectura y *stack* de tecnologías**: Una arquitectura bien definida y la combinación de tecnologías es lo que diferencia principalmente una buena plataforma de otras. Las plataformas pueden estar desarrolladas por un conjunto de tecnologías que son conocidas por no trabajar bien conjuntamente o que dichas tecnologías pueden convertirse en obsoletas pronto, lo cual debe evitarse. Así, la arquitectura debe ser lo suficientemente flexible para futuros cambios, pero al mismo tiempo, ser lo suficientemente rígida respaldada por tecnologías eficientes.
- **Seguridad**: El aspecto de la seguridad es uno de los puntos más importantes de una plataforma IoT debido a que están comprometidos diferentes tipos y cantidades de dispositivos que podrían verse expuestos si no se tiene una buena capa de seguridad. Verificar los *feautures* de seguridad de una plataforma IoT debe ser una obligación. Por ejemplo, se debe brindar seguridad en la capa de transporte (TLS, por sus siglas en inglés *Transport Layer Security*) para comunicarse con los dispositivos y aplicaciones, o la autenticación de dispositivos y usuarios [31].
- **Soporte**: Si bien el soporte continuo para la administración de la plataforma es esencial, también se requiere soporte para fines de integración de la solución. Como requisito obligatorio, la plataforma de *middleware* debe tener un fuerte soporte en el diseño, desarrollo, implementación y administración de la solución de manera continua.

2.3 Plataformas IoT existentes

Hasta el 2021, la cantidad de plataformas IoT reportadas llegaba a ser más de 600 [32], lo cual resulta una tarea retadora analizar cada una de ellas. Sin embargo, en general se pueden dividir a las plataformas IoT en dos categorías: las plataformas en la nube, las cuales usualmente tienen un costo, y las de código abierto, las cuales se pueden descargar y utilizar libremente.

En primer lugar, están las plataformas *cloud*. En [33] se hace un análisis profundo de 26 plataformas *cloud* con la finalidad de analizarlas y encontrar sus puntos a favor y en contra para ser usadas en aplicaciones concretas, como en el desarrollo de aplicaciones, administración de data, investigación, etc. En el análisis realizado se puede encontrar plataformas conocidas como Kaa y ThingSpeak.

Asimismo, en [8] se realizó pruebas con las plataformas Kaa y ThingSpeak enfocadas en la gestión de usuario, el control remoto de un equipo mecánico y las herramientas de visualización que la plataforma ofrece, los cuales mostraron que ninguna plataforma cumplió al 100% con los requerimientos planteados, siendo Kaa la que cumplía con la mayoría de ellas.

Por otro lado, están las plataformas de código abierto. En [34], se realizó un mapeo con el objetivo de encontrar los diferentes tipos de "abertura" que puede tener una plataforma e investigar qué hace que dichas plataformas sean "abiertas". El autor analizó 221 artículos y recolectó las plataformas IoT que más veces hayan sido referenciadas como "abiertas" lo que puede brindar una idea de las plataformas *open-source* existentes que son más nombradas en investigaciones relevantes y en aplicaciones realizadas, ya que se realizó la búsqueda en bases de datos conocidas por tener publicaciones relacionadas al área de IoT como IEEE Explore, ACM Digital Library, SpringerLink, ScienceDirect y Wiley. En la Figura 2.11 se puede ver las plataformas que más veces fueron "usadas" e "indicadas" como abiertas en la bibliografía revisada por el autor en las que se puede encontrar a algunas plataformas conocidas como el ThingsBoard.

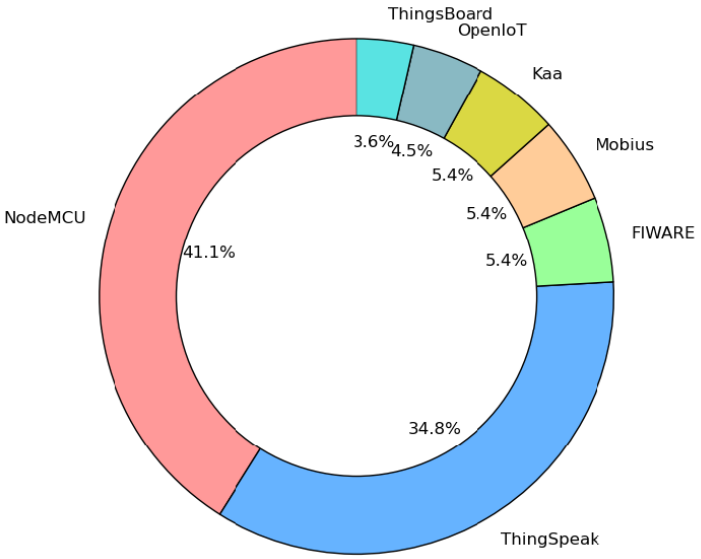


Figura 2.11 Porcentaje de plataformas IoT que son "usados" e "indicados" como abiertas
Fuente: [34]

Asimismo, al igual que con las plataformas IoT *cloud*, en [8] se realizó pruebas utilizando ThingsBoard basadas en los puntos antes mencionados. Al igual que Kaa y ThingSpeak, ThingsBoard no cumplía con todos los requerimientos planteados. Como se puede notar, de las tres plataformas analizadas en [8] ninguna cumplió con todos los requerimientos planteados en su totalidad, por lo que el desarrollo de una plataforma propia pareciese, hasta este punto, la mejor opción. Sin embargo, se analizará las ventajas que supondría el desarrollo de una plataforma IoT frente al uso de una comercial y una de código abierto.

2.4 Comparación de plataformas existentes y la implementación de una propia

Se realizará la comparación entre las dos soluciones propuestas: utilizar una plataforma existente o la implementación propia. Se analizará la personalización, los costos y la escalabilidad de una plataforma, como puntos fundamentales para la elección de la alternativa. Además, se analiza el tiempo de implementación de una plataforma propia, ya que es el argumento, usualmente, más utilizado al momento de optar por una plataforma existente sobre la implementación propia.

2.4.1 Personalización

El principal problema de las plataformas propietarias es la dependencia del proveedor. Es decir, no se pueden realizar modificaciones a la plataforma, en cambio, en muchas de las ocasiones, si no se encuentra una plataforma adecuada para el contexto de una solución, la empresa debe adaptarse a la plataforma y no al sentido contrario. Implementar una plataforma propia brinda la flexibilidad y el control en los requerimientos que se busca cumplir, por lo que la plataforma se adapta a la solución y no la solución a la plataforma [15]. Por otro lado, el optar por una plataforma de código abierto podría ser una mejor opción respecto a las plataformas propietarias, ya que, por definición el *software* de código libre es código fuente que está diseñado para ser accesible públicamente (sin costo), todos pueden analizar el código, modificarlo y distribuir el código como se requiera [35]. Sin embargo, el principal problema con las plataformas de código abierto, y con todo *software* considerado abierto en general, es que usualmente se requiere de un tiempo considerable para modificar un *feature* si es que la plataforma no cumple con las necesidades de una solución en particular. Por ejemplo, ThingsBoard es una plataforma abierta muy conocida y utilizada, pero como se comprobó en [8] no permite la creación de más de un rol diferente al de administrador y no permite el acceso a la plataforma limitado en tiempo, lo cual no permitiría cumplir el requerimiento de brindar acceso a los alumnos basado en fecha y hora a la plataforma (lo que permite que el ingreso de los alumnos se permita solo cuando les toque sus sesiones de laboratorio). Modificar ThingsBoard para que cumpla los dos requerimientos anteriores tomaría tiempo e impediría, además, la actualización de la plataforma si se genera alguna actualización de esta por parte de sus creadores, pues ya ha sido modificada y, probablemente, se generaría una incompatibilidad con la nueva actualización existente. El tiempo utilizado en modificar la plataforma de código abierto existente puede ser invertido en el desarrollo de una plataforma que no necesite de modificaciones, ya que se desarrollaría basado en requerimientos a ser cumplidos desde un

inicio, lo cual lo convierte en una solución altamente personalizable. No obstante, esto último, es subjetivo y depende del grupo de desarrolladores (en este caso, el autor de la tesis) que esté por desarrollar una plataforma u optar por una de código libre. ¿Se prefiere invertir el tiempo disponible en aprender el funcionamiento de la plataforma de código libre y adaptarlo a los requerimientos que se busca cumplir, o implementar una plataforma propia que cumpla, desde el principio, los requerimientos planteados?

2.4.2 Costo y escalabilidad

El costo y escalabilidad están fuertemente relacionadas, ya que a medida que una plataforma escala, los costos aumentan. Las empresas que brindan las plataformas IoT como servicio, usualmente brindan una capa gratuita, por lo que las empresas suelen optar por su uso, ya que es una solución rápida y sin costos. Sin embargo, cuando se necesita de escalamiento se genera dependencia de la plataforma [15]. Por ejemplo, ThingSpeak tiene una capa gratuita que permite la visualización de la data enviada a la plataforma, pero cada 15 segundos, lo cual no permite la visualización de data en tiempo real. Si se desea que la visualización de la data se actualice cada segundo, se necesita pagar una licencia. Así, una mejor opción en términos económicos sería el usar una plataforma de código abierto. Sin embargo, se debe también tener en consideración a la infraestructura *cloud* en la que se despliega la plataforma, la cual también tiene costos. En [31] se realiza una comparación de los costos de una plataforma de código libre desplegada en AWS frente a utilizar una arquitectura de solución utilizando los servidos del mismo proveedor (lo cual sería lo equivalente a utilizar una plataforma propietaria) y se evidencia que a escalas pequeñas las plataformas libres suelen tener un costo de despliegue mayor. Sin embargo, el usar una plataforma *cloud* sigue teniendo la desventaja de no ser personalizable. Por otro lado, la implementación propia apoya a la idea de ser cuidadoso en los

gastos en un inicio y luego invertir solo cuando es necesario, Esto ayuda a aprovechar los avances tecnológicos sin necesidad de presupuestos altos [15].

2.4.3 Tiempo de implementación

Usualmente, las empresas que ofrecen estas plataformas tienen el principal argumento que el tiempo de implementación de una plataforma propia es mayor al tiempo en aprender a usar e integrar la plataforma existente. Sin embargo, no todo desarrollo de una plataforma tiene una duración de años. Hoy en día, con todos los recursos y la tecnología existente disponible, se puede lograr implementar una plataforma en el mismo plazo en el que se toma en aprender la integración de la plataforma de los *vendors* [36] o adaptando la plataforma de código libre al contexto de tu solución.

2.5 Propuesta de implementación propia de plataforma IoT

Por lo expuesto anteriormente, se opta por la implementación de una plataforma IoT propia por los siguientes motivos:

- El principal motivo es que las plataformas analizadas existentes no cumplieron con los requerimientos planteados.
- Utilizar una plataforma existente propietaria que no cumple con todos los requerimientos propuestos significa adaptar los requerimientos planteados a la solución de un *vendor* en particular, lo que generaría dependencia en el proveedor.
- Se considera que adaptar una plataforma IoT de código libre existente de tal manera que cumpla con los requerimientos planteados en 1.4 tomaría más o igual tiempo que desarrollar una plataforma IoT propia que sí cumpla con todos los requerimientos planteados, ya que el desarrollo propio es altamente personalizable.

2.6 Estudio del arte

A continuación, se mostrará diferentes implementaciones de laboratorios remotos realizadas por autores en un contexto educativo que utilicen tecnologías IoT o que su foco sea el ser utilizado por estudiantes. Asimismo, para la revisión de la literatura se buscó soluciones aplicados a equipos que no permitan un control directo por medio de protocolos tcp/ip. Por ejemplo, el control remoto de una PC por medio del protocolo SSH, no fue considerado, ya que el objetivo es controlar un equipo mecánico que por sus propias características no puede conectarse a internet por sí mismo.

2.6.1 Uso de servidores web

En [37], 2017, el autor muestra una arquitectura de solución para laboratorios remotos basada, como lo explica el autor, en las arquitecturas típicas de los laboratorios remotos. Esta arquitectura fue empleada para complementar la enseñanza de los cursos de Sistemas Lineales, Procesamiento de señales y Sistemas de control en la Universidad Complutense de Madrid. La arquitectura propuesta consta de 4 elementos: la interfaz gráfica de usuario (GUI), un servidor, un controlador y la plata.

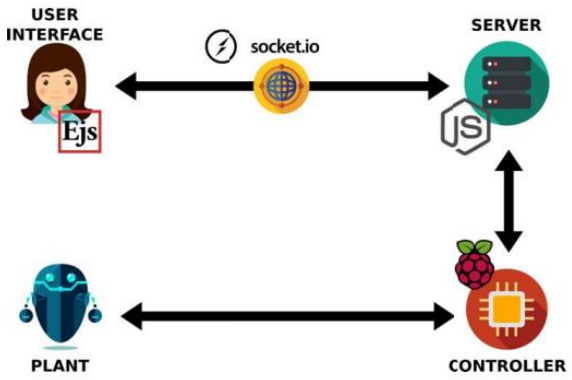


Figura 2.12 Arquitectura típica de un laboratorio remoto
Fuente: [37]

En primer lugar, se tiene a la planta y el controlador. La planta es el equipo/dispositivo físico bajo estudio y el controlador es el encargado de leer la *data* generada por la planta a través de

sensores y controlar a la planta a través de la activación de actuadores. En segundo lugar, se tiene al GUI y al servidor. Por un lado, el GUI es el encargado de mostrar al estudiante el comportamiento de la planta y del controlador; así como permitir la interacción con estos a través del servidor. Para implementar el GUI se hizo uso de interfaces JavaScript embebidos en una página web, los cuales fueron creado sutilizando el software open-source Easy Java/JavaScript Simulations (EJsS). La comunicación entre el GUI y el servidor es a través del protocolo WebSocket. El autor menciona que el uso de WebSockets es óptimo en los laboratorios remotos, ya que reduce la transferencia de data, por el menor tamaño de sus cabeceras, y evita la existencia de *delays*, debido a que existe una comunicación bidireccional y constante. Por otro lado, el servidor es el encargado de proveer acceso al experimento y administra el intercambio de mensajes entre el controlador y el GUI. Se utilizó Node.js junto con Express.js [38], uno de sus más famosos frameworks para aplicaciones web, para implementar un servidor de laboratorio (un servidor web) que permita el acceso seguro al experimento, la redirección de la *data* entre el GUI y el controlador, y mantener un seguimiento de los usuarios del experimento. Asimismo, un Raspberry Pi es utilizado para levantar el servidor y, al mismo tiempo, levantar un controlador basado en C y permitir la comunicación entre ellos. Es decir, en este caso, tanto el servidor y el controlador se encuentras en un mismo dispositivo.

En [39] , 2015, al igual que el caso anterior, se puede encontrar una propuesta que utiliza el protocolo WebSocket. Los autores proponen el uso de dicho protocolo como una alternativa mejorada a la comunicación por HTTP para controlar un brazo robótico remotamente a ser utilizado por estudiantes de escuelas. Los autores utilizaron un Arduino para controlar el brazo robotico y, debido a que se necesita utiliza el protocolo de WebSockets, optaron por utilizar un Raspberry Pi como intermediario, el cual se comunica con el Arduino por medio de un puerto USB. Asimismo, se propuso la implementación de un servidor web en el Rasberry Pi con la

finalidad de que los estudiantes puedan conectarse a este y poder, a través de este, controlar el brazo robótico. Es decir, el estudiante, a través de una interfaz web se comunica con el servidor levantado en el Raspberry y mantiene una comunicación constante por medio de WebSockets. Cuando el Raspberry recibe un mensaje por un canal del WebSocket, este reenvía dicho mensaje al Arduino, por medio del puerto USB, quien, finalmente, controla el brazo robótico. La arquitectura propuesta se observa en la Figura 2.13. Las pruebas realizadas con alrededor de 70 estudiantes, de edad entre los 10 y 18 años, validaron que el control remoto del brazo robótico fue posible.

Cabe resaltar, que al igual que en el caso anterior, Node.js fue utilizado. Sin embargo, inicialmente se planteó dos posibles opciones para lograr el uso de un servidor web, el uso de WebSockets y comunicación con el Arduino. En primer lugar, se tenía a Node,js el cual gracias a sus múltiples librerías ofrecidas por su repositorio NPM se logró obtener librerías que permitan la comunicación entre el Raspberry y el Arduino. La librería elegida fue “serialport” que permitía una comunicación directa entre ambos a través del puerto USB en tiempo real. Por el lado del uso de Websockets, se tenía múltiples opciones. Se optó por la librería Socket.io [40], ya que cuenta con un “respaldo” para su uso en navegadores que no soporten WebSockets, en donde se hace uso de HTTP long-polling. En segundo lugar, se planteó el uso de Python el cual es tradicionalmente utilizado en Raspberrys debido a su simplicidad y a que cuenta con librerías para controlar el *hardware*. Así, se escogió la librería “serial” para el uso del puerto USB. Para el uso de Websockets, se optó por la librería “websocket”. Sin embargo, este no contaba con el “respaldo” que contaba Socket.io. Debido a que se buscaba que la mayor cantidad de estudiantes tengan acceso al laboratorio y, para ese entonces (inclusive en nuestros días, ya que no todos los estudiantes cuentan con la versión más reciente de sus navegadores webs), no todos los navegadores web soportaban WebSockets se eligió el uso de Node.js.

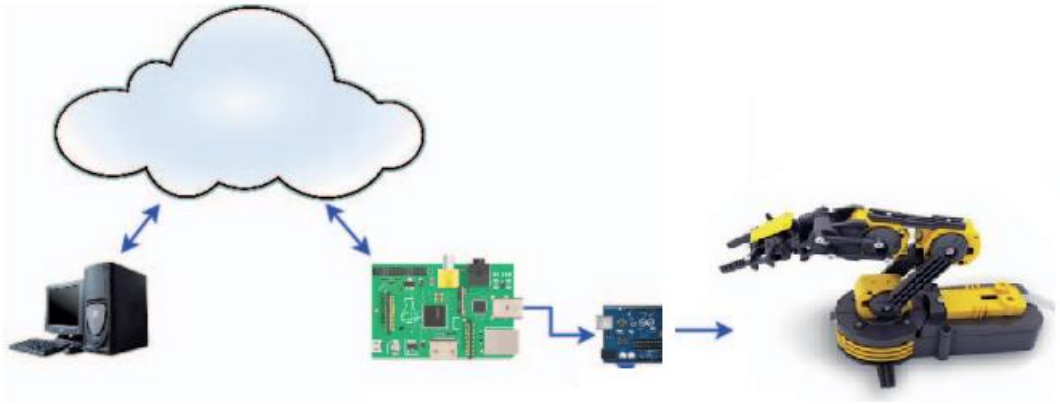


Figura 2.13 Arquitectura propuesta para el control remoto de brazo robótico
Fuente: [39]

Una solución similar más actual, en el 2021, se presenta en [41], donde los autores proponen el diseño de un laboratorio remoto basado en 3 partes: el servidor del laboratorio, la planta a ser controlada y cámaras IP, los cuales pueden verse en la Figura 2.14. El laboratorio remoto propuesto tiene como objetivo controlador un set Lego Mindstorms RCX, el cual ayuda a desarrollar una clase de sistemas de control de retroalimentación.

En primer lugar, el "servidor del laboratorio" es el Raspberry Pi y es el encargado de tener dos servidores corriendo en él, un servidor web y el servidor del experimento. El servidor web permite la interacción del estudiante a través de una GUI en un navegador web, así como el procesamiento de video, la interacción con la base de datos y el proceso de autenticación. Para la implementación de la aplicación web se utilizó el MEAN *stack* (MongoDB, Express.js, Angular.js y Node.js). Por otro lado, el servidor del experimento está encargado de administrar, programar y controlar la planta a través del controlador RCX, el cual es controlado con el lenguaje de programación llamado NQC (*Not Quite C)*. Asimismo, es responsable de la adquisición de la *data* experimental de la planta a ser controlada. En segundo lugar, está la planta a ser controlada, para lo cual se ha utilizado componentes de un antiguo set de Lego Mindstorms para construir una planta de control de ángulo, como un motor DC 9V, un sensor de rotación y engranajes para demostrar el comportamiento del ángulo controlado. Los componentes son controlados a través del controlador antes mencionado. Finalmente, las

cámaras IP se encuentran conectadas al Raspberry Pi 3 para capturar el video en tiempo real de la planta controlada utilizando el protocolo RTSP.

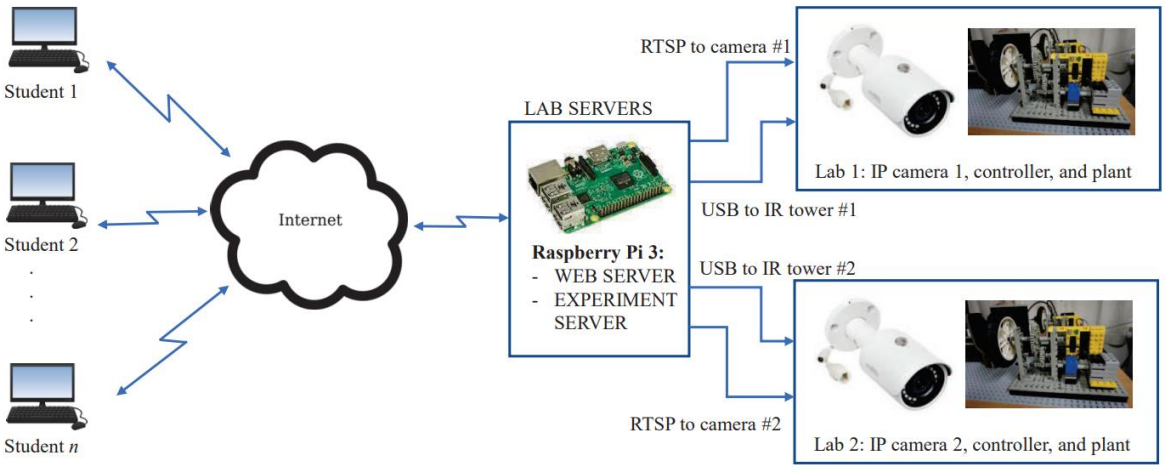


Figura 2.14 Arquitectura propuesta para el control y monitoreo remoto de un set Lego Mindstorms RCX
Fuente: [41]

El laboratorio fue probado con estudiantes que dieron un *feedback* positivo de la experiencia del laboratorio remoto y expresaron que les gustaría tener una variedad de proyectos diseñados; es decir, más laboratorios remotos. En la Figura 2.15, se observa la interfaz desarrollada que cuenta con la imagen de la planta de control de angulo (derecha), una gáfica del ángulo de respuesta (izquierda) y angulos parametros a ser ingresados por los alumnos para interactuar con el laboratorio (debajo de la gráfica).

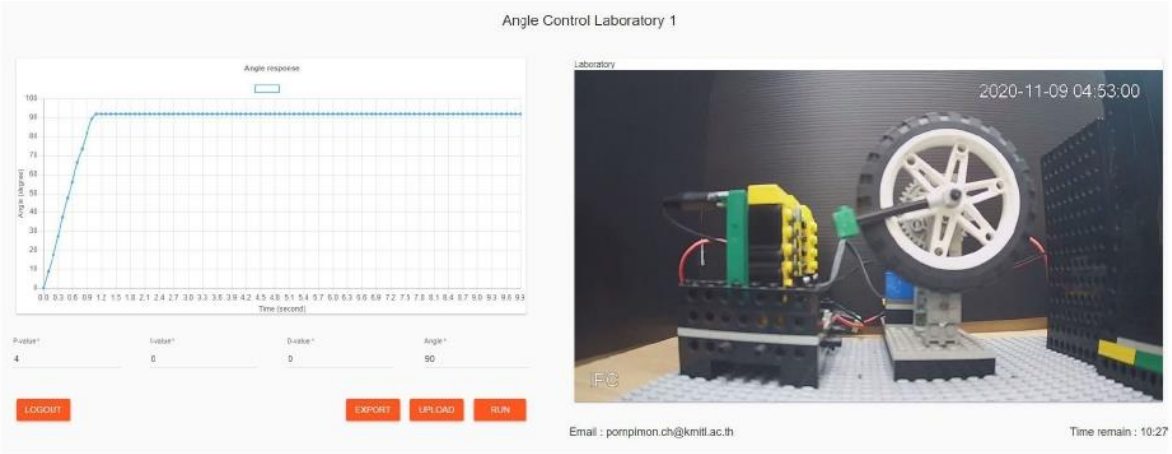


Figura 2.15 Interfaz desarrollada del laboratorio remoto
Fuente: [41]

2.6.2 Uso de Brokers MQTT

Se sabe que el protocolo MQTT es utilizado en aplicaciones IoT debido a que es considerado un protocolo ligero, lo cual es favorable debido a la cantidad de *data* que se maneja en una aplicación IoT, pero también puede ayudar con temas de seguridad. Hasta ahora, se ha mostrado implementaciones en donde el servidor web ha estado levantado en un Raspberry Pi. Sucede un inconveniente con dichas implementaciones: se necesita de una IP pública para poder acceder al servidor corriendo en los Rasbpberry Pi. Es decir, el usuario final se conecta directamente al servidor web, el cual se encuentra en una red interna, que en nuestro caso sería dentro de la universidad. Sin embargo, las instituciones educativas cuentan con políticas de seguridad y procedimientos que restringen el acceso externo a ciertas direcciones IP y puertos para prevenir ataques [42].

En [42], 2021, los autores realizan una validación del uso del protocolo MQTT como una alternativa de solución a ser utilizada por sistemas de laboratorios remotos, en específico, aquellos laboratorios que se encuentran en redes privadas y no puede accederse a ellas a través de una IP pública, como es el caso de las universidades. Para realizar su validación, el autor utiliza una instancia EC2 de AWS con Eclipse Mosquitto MQTT Broker [43] instalado con la finalidad de controlar un laboratorio con una réplica en miniatura de un cruce de ferrocarril. La réplica en miniatura es controlada por un Raspberry Pi 3B+ que controla LEDs (simulando las luces del semáforo) y un servomotor (que controla las rejas de pase de los automóviles). Los autores lograron el intercambio de comandos exitosamente. Asimismo, concluyen que el protocolo MQTT provee una gran flexibilidad, especialmente en entornos de redes privadas.

Se puede notar que el uso del bróker MQTT presenta ventajas respecto al tener un servicio corriendo en algún servidor dentro de una universidad. Otro ejemplo de una propuesta utilizando un bróker MQTT es el que viene a continuación.

En [44], 2021, los autores proponen, como ellos lo llaman, un sistema IoT con la finalidad de mejorar la calidad de la educación en el aprendizaje *online* debido a la pandemia y el cierre de las universidades. Los autores buscan interacción en tiempo real y un uso de ancho de banda pequeña. Para comprobar el funcionamiento del sistema propuesto, realizan el control remoto del encendido y apagado de un motor de inducción trifásico, así como también recibir una señal de control y de *feedback* generado por este mismo.

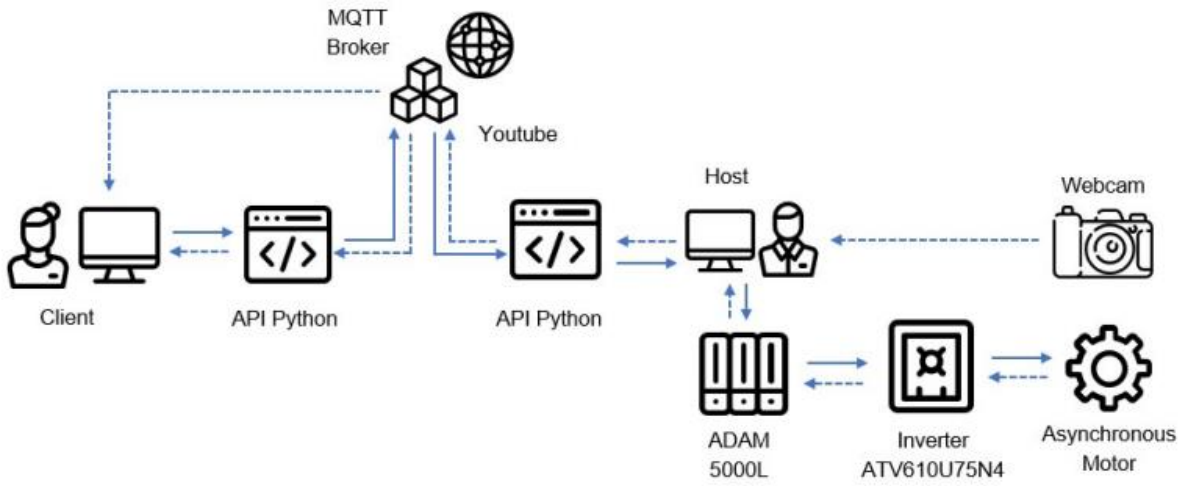


Figura 2.16 Arquitectura de sistema IoT propuesta
Fuente: [44]

El sistema propuesto se observa en la Figura 2.16, en donde proponen el uso de un bróker MQTT, en este caso utiliza Mosquitto MQTT bróker, como la solución más aceptable y económica, ya que es de uso libre y permite la transferencia ligera de data. Los autores concluyen que el uso del protocolo MQTT muestra una buena performance para el desarrollo de laboratorios remotos en tiempo real con un alto porcentaje de confiabilidad del sistema, lo que significa que las probabilidades de fallas son mínimas.

Hasta ahora se ha mostrado el control de equipos pequeños como motores DC. A continuación, se muestra una implementación con equipo mecánicos de un mayor tamaño, por lo que son similares al banco de psicrometría en dicho aspecto, y que no utilizan un bróker MQTT como solución a la no utilización de una IP pública, lo que nos permite analizar que el diseño de una solución depende también del contexto en donde se implemente.

En [10], 2021, los autores proponen la implementación de una plataforma IoT para el monitoreo de clústeres de paneles fotovoltaicos ubicados en el Laboratorio de Energía Solar de la Universidad de Filadelfia, Jordania. La plataforma propuesta está conformada por un el conjunto de clústeres, un controlador, un servidor local y un servidor web por el cual los estudiantes pueden conectarse desde cualquier lugar conectados a internet.

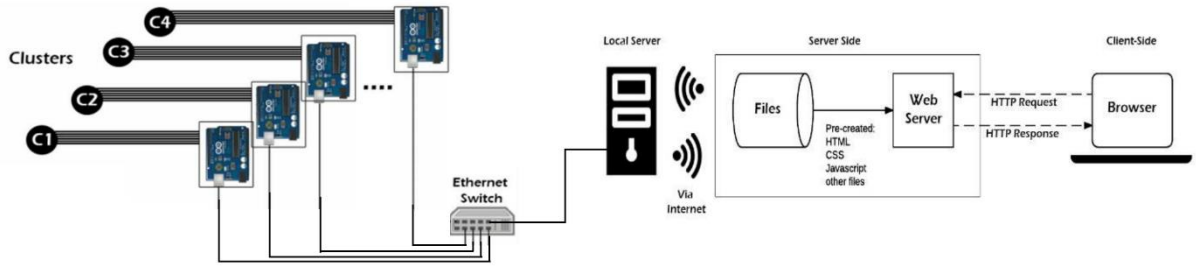


Figura 2.17 Arquitectura de la plataforma IoT propuesta para el monitoreo de clústeres de paneles fotovoltaicos
Fuente: [10]

Cada clúster cuenta con cuatro paneles fotovoltaicos y cada panel está equipado de sensores para medir el voltaje, corriente e irradiancia. Asimismo, cada clúster cuenta con sensores de temperatura y humedad. De igual manera, cada clúster está conectado a un controlador, el cual es un Arduino UNO de 14 pines digitales I/O y un *shield Ethernet*. El controlador es usado para escanear los parámetros medidos por los sensores y, cada cierto tiempo, según lo configuren, enviárselos al servidor local a través de un cable Ethernet. Una vez la *data* se encuentre en el servidor local, este se lo envía al servidor web a través de un POST HTTP. El servidor web lo almacena en un servidor MySQL para su persistencia. Cuando un estudiante accede al servidor web, el servidor muestra la *data* utilizando el lenguaje de programación PHP, HTML y CSS para el GUI. Asimismo, para que el estudiante tenga acceso al servidor web, se asoció a este un nombre de dominio, en este caso fue “www.philadelphia.edu.jo”.

Podemos notar que, en esta última implementación se utiliza un servidor web intermediario que permite la conexión de los estudiantes a través de internet, a diferencia de la propuesta anterior en donde se propone un bróker MQTT para no depender de una IP pública o de las

implementaciones de conexión directa con el servidor web dentro de la institución educativa. Asimismo, cabe resaltar que las pruebas realizadas utilizaron un tiempo de censado de *data* de 30 minutos. Así, se puede concluir, que las ventajas de utilizar un bróker MQTT, sobre todo él envió de *data* en tiempo real, no sería necesario.

2.6.3 Síntesis de las implementaciones analizadas

En síntesis, de las implementaciones mostradas podemos notar que el uso de Arduinos y Raspberrys es común en laboratorios remotos y se les conoce como “controladores”, ya que controla los sensores y actuadores del equipo a monitorear y controlar. Asimismo, se ha evidenciado que el factor web es muy importante en la capa de aplicación, ya que el uso de un servidor web es muy utilizado para permitir a los estudiantes acceder y controlar el laboratorio remotamente a través de una interfaz web. Asimismo, hemos hallado que el uso de un bróker MQTT, debido a que tiene una cabecera de menor tamaño en comparación con HTTP, es más conveniente utilizarlo en situaciones donde el flujo de *data* es grande y en intervalos pequeños. Para situaciones donde no sucede esto último, podría utilizarse HTTP sin inconvenientes, como se mostró en [10].

Capítulo 3. Propuesta de diseño y arquitectura

3.1 Requerimientos de la plataforma

A continuación, se listan los requerimientos funcionales necesarios para el desarrollo de la plataforma:

- **R1. Gestión de usuarios basado en roles:** El desarrollo de un laboratorio dentro de la PUCP tiene involucrado a tres involucrados. Los alumnos, que son los que desarrollan el laboratorio; el jefe de práctica (JP), encargado de guiar a los alumnos en el desarrollo de sus experiencias; y el administrador, encargado de asignar coordinar temas de coordinación de acceso al laboratorio, generalmente este rol será tomado por la secretaria del laboratorio.
- **R2. Acceso a la plataforma a través de credenciales o a través del correo PUCP:** Por temas de seguridad, se requiere que los estudiantes puedan acceder a la plataforma con unas credenciales que validen que son realmente los estudiantes que tienen asignados un laboratorio y, además, que el acceso solo se permita cuando tenga asignado una sesión de laboratorio. Asimismo, que puedan utilizar sus cuentas PUCP para poder

acceder, ya que cada estudiante tiene asignado su propio correo PUCP, lo que facilitará el acceso a la plataforma.

- **R3. Automatización del acceso a la plataforma por los alumnos basado en fecha y horas:** Debido a que las fechas en las cuales un alumno tendrá su laboratorio están predefinidas desde el inicio de un ciclo académico, se considera que es necesario que los estudiantes tengan acceso a la plataforma de manera automática en las fechas prefinidas, sin necesidad de configurar los accesos de los estudiantes cada vez que inicie un nuevo laboratorio.
- **R4. Variables a controlar:** Se tiene tres tipos de variables a controlar. En primer lugar, se tiene las variables de entrada, las cuales son las que el banco de psicrometría envía a la plataforma. Se tiene 21 variables de entradas que enviarán *data* cada segundo a la plataforma, lo que significa que se tiene 21 mensajes por segundo que el equipo generará y será recibida por la plataforma. En segundo lugar, se tiene a las variables de salida, las cuales son que son las que se envían desde la plataforma. Se tiene 10 variables de salida. Finalmente, se tiene variables de video, los cuales se explica en el siguiente punto.
- **R5. Asignación por parte de los jefes de práctica de los interruptores que pueden ser controlados por un alumno en tiempo real**: En el desarrollo de un laboratorio en donde se utilice el banco de psicrometría, los alumnos necesitan interactuar con el equipo y con los interruptores que este tiene para el desarrollo de sus experiencias. Por ende, el JP debe poder asignar qué interruptores podrá ser controlado por un estudiante en particular. Adicionalmente, se necesita que dicho control sea en tiempo real para mantener el dinamismo de la experiencia.
- **R6. Visualización en tiempo real de los alumnos conectados**: Debido a que el laboratorio se desarrolla de manera no presencial se considera necesario que el JP tenga

una manera de controlar qué alumnos se encuentras realmente conectados a la plataforma para facilitar las coordinaciones con los estudiantes.

- **R7. Visualización del comportamiento de la data generada por el equipo en tiempo real**: En el desarrollo de un laboratorio convencional, los estudiantes tienen que realizar mediciones en el equipo para el desarrollo de sus experiencias. Por lo cual, se propone que el sistema permita visualizar la evolución de los parámetros a monitorea a través de un *dasboard*. Asimismo, dicha visualización deberá ser en tiempo real para una mejor experiencia de usuario y permitir que el estudiante pueda detectar cualquier cambio que suceda con algún parámetro y realizar el análisis correspondiente.
- **R8. Control remoto del banco de psicrometría**: El alumno podrá controlar los interruptores del banco de psicrometría en tiempo real a través de la interfaz web. Asimismo, la acción de algún estudiante debe reflejarse en la interfaz del resto de usuarios conectados (alumnos, jefes de prácticas y administradores) en tiempo real.
- **R9. Visualización de medidores que no generen *data* (como un nanómetro) mediante cámaras**: Existen variables que no es posible la recolección de data por medio de sensores debido a sus limitaciones mecánicas. Por lo que se propone que dichas variables puedan ser visualizadas mediante cámaras colocadas en la experiencia. Se les considera variables de video.
- **R10. Concurrencia de usuarios:** La cantidad de usuarios concurrentes es de 8 aproximadamente; 6 alumnos en promedio, 1 JP y 1 administrador. Esto debido a que, según personal del laboratorio y como se puede observar en el Anexo 1, los alumnos se agrupan en grupos entre 4 a 6 estudiantes.
- **R11. Descarga de *data* de los ensayos.** Se propone que los estudiantes puedan descargar un archivo CSV con los valores generados por el banco de psicrometría de todas las variables.

- Al desarrollo de un laboratorio se le conoce como “ensayo” y a los ensayos en donde se utiliza el banco de psicrometría se le conoce como “ensayo de psicrometría”

3.2 Arquitectura de la solución

Una vez descritos los requerimientos se propone la arquitectura de solución mostrada en la Figura 3.1.

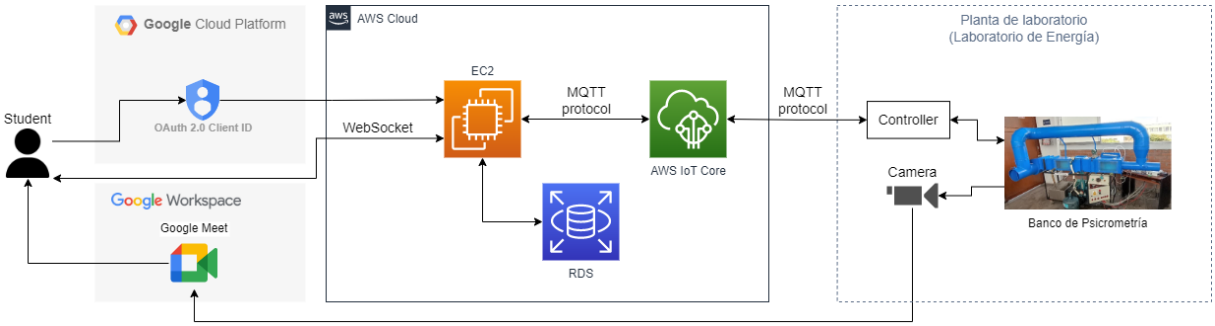


Figura 3.1 Arquitectura de solución propuesta
Fuente: Elaboración propia

A continuación, se describirán los 4 módulos propuestos en la arquitectura de solución:

- Google Cloud Platform
- Google Workspace
- AWS Cloud
- Planta de laboratorio

3.2.1 Google Cloud Platform

El módulo Google Cloud Platform está conformado por los servicios o configuraciones realizadas en la nube de Google. Se propone que la autenticación de los usuarios sea de dos maneras. En primer lugar, se propone utilizar el protocolo OAuth 2.0 utilizando a Google como el *authorization server*. Para lograr esto último, se debe registrar la aplicación en la consola de Google Cloud y generar credenciales OAuth 2.0. En segundo lugar, se permitirá ingresar con

unas credenciales propias del alumno (correo y contraseña) las cuales se almacenarán en el RDS.

3.2.2 Google Workspace

Se propone el uso de Google Meet de Google Worksplace para la visualización de las cámaras colocadas con la finalidad de que el alumno monitoree variables que no cuenten con sensores, como se explicará en 4.1.2.

3.2.3 AWS Cloud

El módulo AWS Cloud está conformado por los servicios a ser utilizados de la nube de AWS. Asimismo, dichos servicios serán los bloques que conformen IoT middleware.

3.2.3.1 AWS IoT Core

Los sensores con los que cuenta el banco de psicrometría se encontraran constantemente obteniendo *data* y es necesario que dicha data llegue a la plataforma mediante un protocolo de comunicación. Los dos protocolos más utilizados en las soluciones IoT son el CoAP y MQTT, cada uno con sus ventajas y desventajas [17]. Ambos protocolos están diseñados para ser utilizados por dispositivos limitados computacionalmente, por lo que pueden ser utilizados por Arduinos y Raspberrys (como el caso mostrado en [42]), y un menor consumo del ancho de banda, ya que cuentan con cabeceras pequeñas (CoAP con 4 bytes y MQTT con 2 bytes [17]). Sin embargo, el protocolo que más se adecua a la solución propuesta en la presente tesis es MQTT, debido a su arquitectura de suscripción y publicación, ya que permite comunicaciones del tipo *many-to-many* [19]. El protocolo CoAP, debido a su arquitectura *request*/*response* y a su compatibilidad con HTTP, es un protocolo orientado a una comunicación *one-to-one* [19] lo cual no sería conveniente utilizar por temas de escalabilidad si se planea utilizar la solución para otros equipos además del banco de psicrometría. Por su lado, el protocolo MQTT, permite

que múltiples clientes puedan enviar mensajes al bróker, que es el punto central, y este es el que decide a quien envía dicha *data*. Además, MQTT está orientado a la conexión (*connection-oriented*) [17]. Es decir, la conexión entre el cliente y el *broken* se mantiene abierto, lo cual permite una comunicación bidireccional en tiempo real que es requisito en nuestra solución. Por su lado CoAP, necesita crear una nueva conexión cada vez que se necesite envío de *data*, lo cual puede generar cierto *delay*, ya que la frecuencia de envío de *data*, en nuestro caso, es 21 mensajes por segundo, considerando solo las variables de entrada que son la *data* constante que el equipo estará enviando. Sin embargo, esto conlleva a un mayor consumo de energía. En un ambiente IoT puro, es usual que los dispositivos busquen ser lo más eficientes posible con el consumo de energía; sin embargo, en nuestro caso no es un factor a tener en cuenta, ya que el dispositivo se encontrará dentro de la universidad y contará con un suministro de energía constante. Así, el protocolo elegido es MQTT.

Para implementar el broker MQTT se propone el uso del servicio AWS IoT Core de AWS, el cual provee una comunicación segura y bidireccional entre dispositivos conectados a internet y la nube de AWS sobre MQTT, HTTPS o LoRaWAN [45]. Para la presente solución se usará una comunicación sobre MQTT por lo anteriormente desarrollado.

3.2.3.2 RDS (Amazon Relational Databases)

La plataforma IoT a implementar necesitará una capa de persistencia debido a los requerimientos listados en 3.1, ya que es necesario guardar los datos de los usuarios para poder gestionarlos, asignación de accesos según los roles, almacenamiento de credenciales, almacenamiento de las fechas y horas en las que a un alumno tiene asignado un laboratorio, almacenamiento de las variables que un alumno en específico puede controlar y almacenamiento de la data generada por los sensores que monitorearan al banco de psicrometría. Como se mencionó en 2.1.5, existen dos posibilidades de elección para la capa de persistencia: las bases de datos relacionales (o SQL) o las NoSQL. Para la elección del tipo de

base de datos se analizará qué tipo de *data* se necesita almacenar y de qué manera esta se almacenará. Para ello, se divide la *data* a ser almacenada de acuerdo a los requerimientos, anteriormente listados, la cual puede ser observada en la Tabla 3.1. De dicha división se puede observar que la información a almacenar tendrá relaciones entre ellas. En primer lugar, los usuarios, para el caso de los de rol alumno, tienen horarios asignados a los cuales pertenecen. En segundo lugar, cada ensayo tiene una cantidad de variables, sean de entrada o de salida, relacionadas a cada ensayo. Por último, cada usuario tiene asignado cierto ensayo en una fecha predefinida y, de igual manera, una cierta cantidad de variables asignadas que dicho usuario puede controlar. Asimismo, se puede notar que la *data* almacenada puede tener una estructura definida y dicha estructura no necesita de ser dinámica. Por ejemplo, cada usuario a almacenar tendrá las mismas características (su rol y sus datos personales) y no se tendrá el caso en el que se necesite que un usuario almacene su número de DNI (Documento Nacional de Identidad).

Tabla 3.1 Análisis de la data a ser almacenada
Fuente: elaboración propia

Funcionalidad	Entidades	Descripción
Gestión de usuarios	Usuario	Representa a los usuarios que utilizarán la plataforma. Se almacenará el rol del usuario (alumno, JP o administrador) y sus datos personales (nombres, apellidos, código de estudiante, correo, contraseña).
	Horario	Representa al horario al cual estudiante pertenece.
Gestión de ensayos	Ensayo	Representa al ensayo a realizarse. Si se desea que la plataforma pueda ser utilizado por varios equipos dentro del laboratorio de Energía, se necesita esta entidad para separar cada ensayo.
	Variable de ensayo	Representa a las variables que tiene cada ensayo, Se almacenará el nombre de la variable, el tipo (puede ser de entrada, del equipo a la plataforma, o de salida, de la plataforma al equipo) y el *topic* MQTT relacionado a la variable.

Así, se puede observar que la *data* a almacenar tiene relaciones entre cada una de ellas y pueden

ser almacenadas con una estructura definida, por lo que una base de datos relacional es la que más se adecua. Además, las propiedades ACID que brinda una RDBMS permitirán tener un buen funcionamiento en el desarrollo de un laboratorio, ya que garantizarán, por ejemplo, que la asignación de un conjunto de variables a un estudiante en específico se logre correctamente. Sería un problema que dos estudiantes, por algún error en la base de datos, tengan la misma variable asignada, lo cual produciría problemas de coordinación en el laboratorio o, incluso, causar daños en el equipo físico por mala manipulación de este.

Sin embargo, las bases de datos no relacionales son usualmente utilizadas en soluciones IoT, ya que ejecutan operaciones más rápidas para cantidades de volúmenes grandes [27], pero, como se pudo observar, no necesariamente siempre se adecuan al tipo de dato a almacenar, como en nuestro caso. No obstante, cabe tener en cuenta que los valores que llegarán a la plataforma de los sensores del equipo, considerando solo los de entrada, serán aproximadamente 21 escrituras por segundo en base de dato y las bases de datos NoSQL demuestran un mejor *performance* en operaciones de escritura que las relacionales [26]. Se considera que para un equipo (el banco de psicrometría), este aspecto no tendrá un gran impacto, pero a medida que se aumenten más ensayos se podría presentar algunos problemas. La ventaja de la implementación propia es que se puede ir agregando *features* a una solución a medida que esta va creciendo. Así, se podría implementar un sistema de bases de datos híbrida, las cuales muestran una mejor performance en aplicaciones web en donde se tenga una alta cantidad de escrituras [46] y mejorarían la performance en aplicaciones IoT, de igual manera. La RDBMS se encargaría del almacenamiento de la gestión de usuarios y la gestión de ensayos, debido a la notable dependencia de relaciones entre la *data* almacenada, y la base de datos NoSQL se encargaría de almacenar únicamente los valores recibidos de los sensores en los equipos. Sin embargo, debido a que la presente tesis se implementará para el banco de psicrometría, se opta solo por el uso de una RDBMS.

De las RDBMS existentes, se elige MySQL debido a que es conocido por su alta velocidad de ejecución de *querys* en comparación con otras RDBMS, como PostgreeSQL u Oracle [46].

Para la implementación de MySQL se propone el uso del servicio de Amazon RDS de AWS. Amazon RDS es un servicio gestionado de base de datos que brinda automatización en la configuración y mantenimiento de una base de datos relacional, tareas que pueden ser tediosas en el desarrollo de una solución. Asimismo, AWS se encarga de aprovisionamiento de hardware, implementación de parches, creación de *backups*, etc [47].

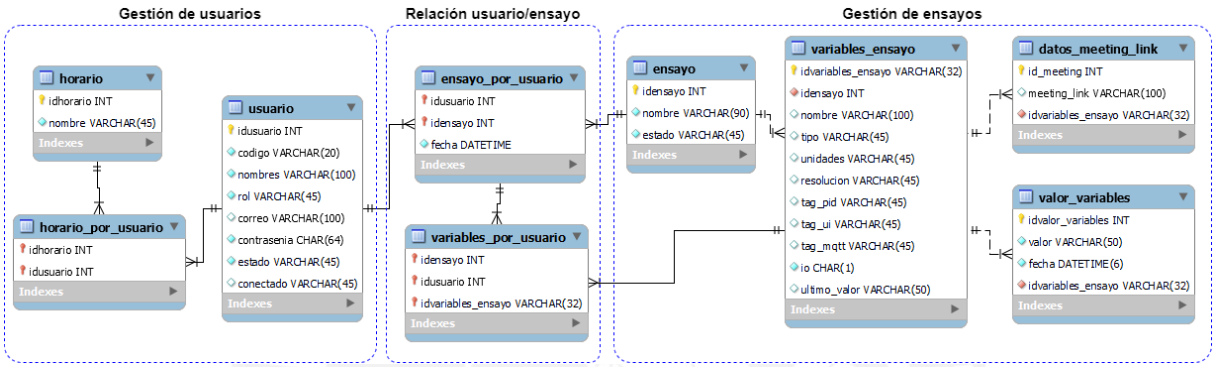


Figura 3.2 Diseño de esquema entidad-relación
Fuente: Elaboración propia

Finalmente, se propone el esquema entidad-relación de la base de datos a implementar mostrada en la Figura 3.2, la cual se ha implementado basado en la gestión de usuarios y gestión de ensayos mostradas en la Tabla 3.1. Con el esquema entidad-relación propuesto se logra las funcionalidades listadas en la Tabla 3.2.

Tabla 3.2 Funcionalidades obtenidas del esquema entidad-relación propuesto
Fuente: Elaboración propia

Funcionalidad	Tabla	Descripción
Gestión de usuarios	usuario	Representa a la entidad usuario de la Tabla 3.1. Se observa que se tiene el atributo "rol", el cual almacenará si el usuario es alumno, jefe de practica o administrador (para cumplir el requerimiento R1 de gestión de usuarios por roles).
	horario	Representa a la entidad horario de la Tabla 3.1.

	horario_por_usuario	Representa la relación entre la tabla *usuario* y *horario*, ya que algunos usuarios (alumnos y jps) tendrán asignados un horario en particular.
Gestión de ensayos	ensayo	Representa a la entidad ensayo de la Tabla 3.1.
	variables_ensayo	Representa a la entidad variable de ensayo de la Tabla 3.1. Se observa que se tiene el atributo "io", el cual almacenará el tipo de variable (si es de entrada, de salida o de video) para cumplir el requerimiento R4 (existencia de varios tipos de variables).
	valor_variables	Tabla que almacenará la evolución de las variables. Se almacena el valor y la fecha en la que dicho valor fue enviado a la plataforma.
	datos_meeting_link	Tabla que almacenará el enlace de la sesión *meet* de una variable tipo video.
Gestión de relación ensayo/usuario	ensayo_por_usuario	Representa la relación de la tabla usuario y ensayo. Almacena el usuario, el ensayo que le corresponde y la fecha en la cual tiene asignado dicho ensayo.
	variables_por_usuario	Representa la relación de la tabla usuario y variables_ensayo. Almacena el usuario, el ensayo y la variable asignada que podrá ser controlada por dicho usuario.

3.2.3.3 EC2 (Amazon Elastic Compute Cloud)

El servicio Amazon Elastic Compute Cloud provee capacidad computacional escalable y permite escalar hacia arriba o hacia abajo según los requerimientos de una aplicación. EC2 permite crear servidores virtuales, configurar su seguridad y el *networking*, y gestionar su almacenamiento [48]. Así, se propone el uso de EC2 para crear un servidor virtual e implementar un servidor web en este.

a. Servidor Web Node.js con Express.js framework

Se implementará un servidor web para permitir a los usuarios (estudiantes, JP y administradores) interactuar con la plataforma IoT a través de una interfaz web. Existen muchos lenguajes de programación y cada uno con sus propios *frameworks* que facilitan la

implementación de un servidor web. Sin embargo, un criterio de elección será la tecnología a implementar en el servidor web. Dicha tecnología es el protocolo WebSocket. Debido a que Node.js muestra un soporte óptimo del protocolo WebSocket, como se pudo observar en el punto 2.1.4, se propone su uso para la implementación del Servidor Web. Asimismo, se propone el uso de la librería Express.js [38], ya que es uno de sus más famosos *frameworks* para aplicaciones web.

b. WebSockets

Se propone el uso de WebSockets en el Servidor Web, ya que permitirá el envío de *data* a los usuarios conectados en tiempo real y el cumplimiento de los requerimientos que requieren de una tecnología de tiempo real como la asignación de variables en tiempo real, visualización del estado de los alumnos en tiempo real y visualización de la *data* generada por el equipo en tiempo real. El uso de WebSocket permitirá tener una comunicación bidireccional, lo cual permite la comunicación en tiempo real, y un consumo menor del ancho de banda, ya que no se realiza el envío de cabeceras HTTP constantemente como en el método *HTTP polling* y *HTTP long polling*.

Se propone el uso de la librería Socket.io [40], ya que cuenta con un *fallback* de *long-polling,* lo cual permitirá que la mayor cantidad de navegadores puedan utilizar la solución propuesta, como se demostró en la propuesta desarrollada en [39]. Esto permitirá que usuarios con navegadores que no soporten el protocolo WebSocket puedan utilizar la plataforma. Así, se puede lograr que la solución no solo sea utilizada por estudiantes de la PUCP, sino por la mayor cantidad de estudiantes posible como se plateó en 1.7.2.

3.2.4 Planta de laboratorio

El módulo planta de laboratorio está conformado por los elementos que se encuentran físicamente en el Laboratorio de Energía de la PUCP. En primer lugar, se tiene al controlador,

siguiendo la terminología de la una arquitectura típica de un laboratorio remoto propuesta en [37]. El controlador será el encargado de recolectar la *data* y enviarlo a la plataforma. Inicialmente, el controlador era solo un Arduino Mega, el cual tenía los sensores conectados a él. Sin embargo, debido a que no podía realizar una comunicación segura, por medio del protocolo TLS, por sus limitaciones de hardware, se propone la adición de un Raspberry Pi, conectado al Arduino, y sea este quien realice la conexión con AWS IoT Core. El Raspberry se conecta al Arduino por medio de un puerto serial, como se verá en 4.1.3.3. En segundo lugar, se tiene el banco de psicrometría, el cual tiene los sensores conectados a él. Finalmente, el laboratorio propuso el uso de celulares para capturar aquellas variables que no cuentan con sensores, como los nanómetros, para que el estudiante pueda observar cómo varían dichas variables a través de las cámaras de dichos celulares. Asimismo, se cuenta con una cámara IP que brinda una vista panorámica del laboratorio, para que los alumnos puedan ver cómo el equipo va reaccionando a las acciones que ellos realizan.

3.2.4.1 Variables propuestas

Figura 3.3 Zonas en el banco de psicrometría
Fuente: elaboración propia

El banco de psicrometría condiciona aire mediante el aumento de temperatura, disminución de temperatura, aumento de humedad, disminución de humedad, etc. Así, se puede analizar el proceso de aire. Se puede separar el equipo en zonas, como puede observarse en la Figura 3.3.

En la zona A, se tiene aire a temperatura ambiente, el cual ses succionado por un ventilador. Luego, el aire aumenta de temperatura debido a una transferencia de calor de dos resistencias, las cuales son calentadas por corriente de 220V. Así, en la zona B, se tiene aire con una temperatura mayor a la inicial.

En la zona C, se ha aumentado la humedad del aire debido a que se ha realizado una inyección de vapor, el cual es generado por el calentador e ingresa por una manguera. El calentador cuenta con resistencias de inmersión internas que transfieren calor a una cantidad de agua que se encuentra en dicho calentador. El agua se evapora y el vapor generado es el enviado por la manguera.

En la zona D, la temperatura del aire ha disminuido debido a que se transfiere calor a un refrigerante del sistema de refrigeración. Para poder calcular cuánta temperatura ha sido transferida al sistema de refrigeración y para calcular otros cálculos, como la eficiencia, que son parte de los laboratorios, se tiene que medir temperaturas y presiones en diferentes puntos de dicho sistema.

En la zona E, se vuelve aumentar la temperatura debido a que existe resistencias al igual que las que existe entre las zonas A y B.

Finalmente, se puede observar un cambio de la sección del recorrido del flujo del aire, lo que ocasiona un cambio en la presión, existe un “diferencial de presión”.

Dado el flujo del aire por todo el equipo del banco de psicrometría, los ingenieros Diego Pichilingue y Arturo Berastain proponen variables que representen las temperaturas, humedades, corrientes, tensiones y presiones de todo el proceso, las cuales se listan en la Tabla 3.3 y en la Tabla 3.4.

Tabla 3.3 Variables de entrada
Fuente: elaboración propia

Variable	Descripción
Temperatura ambiente – Zona A, B, C, D, y E	Representa las temperaturas en las zonas A, B, C, D y E.
Humedad ambiente – Zona A, B, C, D, y E	Representa las humedades en las zonas A, B, C, D y E.
Corriente eléctrica - Resistencia 1, 2, 3 y 4	Representa a la corriente de las resistencias existentes entre la zona A/zona B y zona D/ zona E. Se tiene pares de resistencias en paralelo. Es decir, entre zona A/zona B se tiene 2 resistencias con corrientes 1y 2, y en zona D/ zona E se tiene 2 resistencias más con corrientes 3 y 4.
Nivel de agua - calentador	Representa la existencia o no de agua en el calentador.
Temperatura refrigerante - Estado 1, 2, 3 y 4	Representa las temperaturas en los cuatro puntos a analizar del sistema de refrigeración.
Presión diferencial - Placa orificio	Representa el cambio de presión debido al cambio de la sección del recorrido del flujo del aire.
Tensión eléctrica - General	Representa la tensión general que ingresa al sistema.
Fin de carrera	Representa el tope de máximo y mínimo del control de velocidad del motor

Tabla 3.4 Variables de salida
Fuente: elaboración propia

Variable	Descripción
Encendido general	Representa el botón de encendido general del banco de psicrometría.
Encendido ventilador	Representa el *switch* de encendido del ventilador que succionar el aire en el punto inicial.
Motor de Pasos/Variac	Representa al *variac* que controla cuanto voltaje tendrá el ventilador de la zona A (el control de velocidad del motor).

Encendido compresor	Representa el *switch* de encendido del compresor que enfría el refrigerante del sistema de refrigeración
Encendido resistencia 1, 2, 3 y 4	Representa el *switch* de encendido de la corriente eléctrica en las resistencias para que estás se "calienten".
Encendido resistencia A, B y C	Representa los *switches* de encendido de las resistencias de inmersión que se encuentran dentro del calentador.

Asimismo, como se mencionó anteriormente, existen variables que no cuentan con sensores, como las presiones del sistema de refrigeración, ya que dicho sistema es un sistema hermético y resultaría muy difícil adaptarlo con un sensor, ya que el banco de psicrometría es un equipo antiguo que ya no se encuentra en producción, por lo que si alguna pieza queda inutilizable deberá ser fabricada por el mismo Laboratorio de Energía.

Tabla 3.5 Variables de video
Fuente: elaboración propia

Variable	**Descripción**
Presión refrigerante - Estado 1, 2, 3 y 4	Representa las presiones del sistema de enfriamiento.
Flujo de refrigerante - Rotámetro	Representa el caudal másico del refrigerante
Temperatura condensado - Probeta	Representa la temperatura del líquido condensado que sale del proceso
Volumen condensado - Probeta	Representa del volumen del líquido condensado que sale del proceso
Paneo general	Representa a la cámara de paneo del laboratorio.

Capítulo 4. Implementación y pruebas realizadas de la plataforma IoT propuesta

4.1 Implementación de la planta de laboratorio

Se presenta la implementación del módulo de planta de laboratorio, el cual está conformado por los elementos que se encuentran físicamente en el Laboratorio de Energía de la PUCP: el controlador, el banco de psicrometría y las cámaras. Se mostrará la implementación de sensores y actuadores conectados al equipo físico, de las cámaras utilizadas y del controlador, al cual se conectarán los sensores y actuadores.

4.1.1 Sensores y actuadores

Se desplegó sensores para obtener la data de las variables de entrada y actuadores para las variables de salida. Los ingenieros Pichilingue y Berastain fueron los encargados del análisis, elección y colocación de los sensores y actuadores en el banco de psicrometría. Así, se utilizaron los sensores listados en la Tabla 4.1 para cada variable propuesta anteriormente.

Tabla 4.1 Sensores utilizados
Fuente: elaboración propia

Variable	Sensor utilizado
Temperatura ambiente – Zona A, B, C, D, y E	Sensor de humedad relativa y temperatura DHT22
Humedad ambiente – Zona A, B, C, D, y E	
Corriente eléctrica - Resistencia 1, 2, 3 y 4	Sensor de corriente ACS712
Nivel de agua - calentador	Medidor de nivel por flotador magnético
Temperatura refrigerante - Estado 1, 2, 3 y 4	Termopar y transmisor MAX6675
Presión diferencial - Placa orificio	Yokogawa EJX110A
Tensión eléctrica - General	Medidor de voltaje Circutor
Fin de carrera	Pulsador

Los sensores y actuadores están unidos al Arduino, el cual se encarga de adquirir la data de los sensores y activar/desactivar los actuadores. En la Figura 4.1 se observa cómo están conectados los sensores al Arduino.

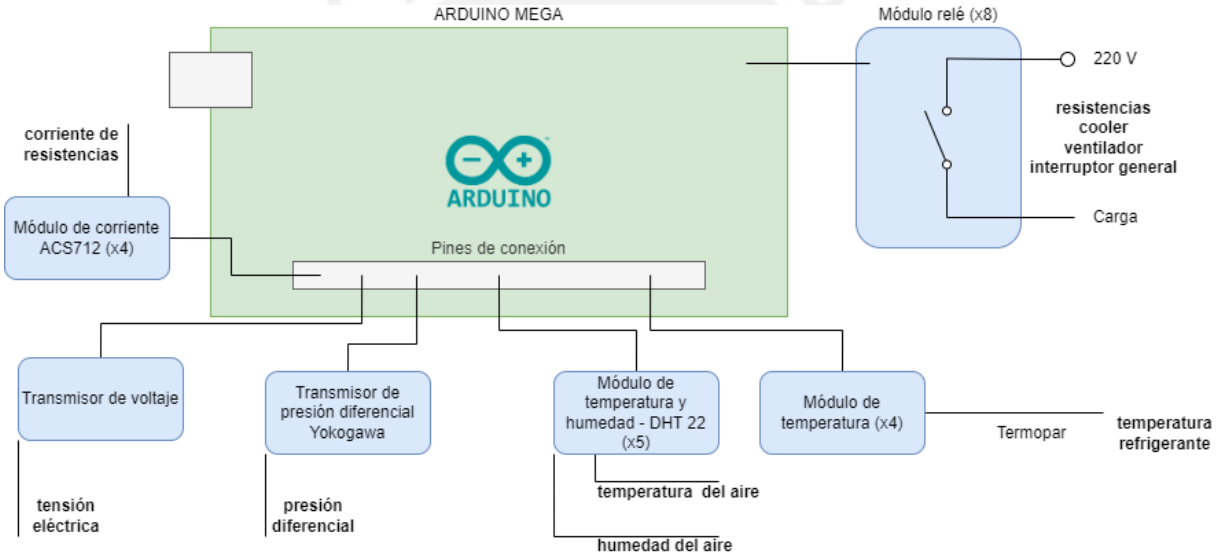


Figura 4.1 Esquema de conexiones entre el Arduino y sensores
Fuente: elaborado propia en colaboración con el Ingeniero Diego Pichilingue

Asimismo, se utiliza relés conectados a pines digitales del Arduino. Un relé se encarga de indicarle a una bobina cuándo debe cerrarse energizándolo con 5V. Así, se controla eléctricamente un *switch*, como se observa en la Figura 4.2. Los relés se utilizan para controlar el encendido general, encendido de ventilador, encendido compresor y encendido de resistencias (1, 2, 3, 4, A, B, y C).

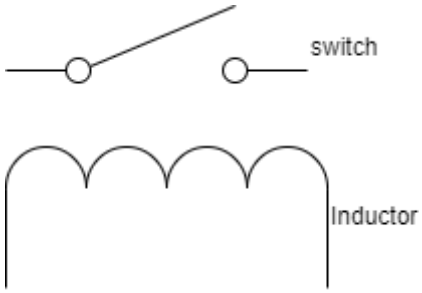


Figura 4.2 Esquema eléctrico de un relé
Fuente: elaboración propia

Por último, para controlar el motor de pasos se utiliza un módulo de controlador, el cual recibe una señal serial de control para que avance o retroceda el motor (*stepper*) que se encuentra conectado a dicho módulo.

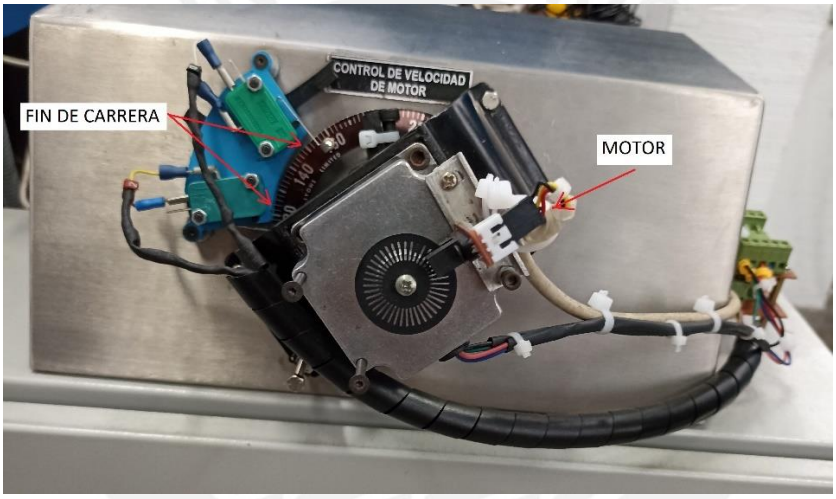


Figura 4.3 Stepper y fin de carreras implementados por Pichilingue y Berastain
Fuente: elaboración propia

4.1.2 Cámaras

Tabla 4.2 Especificaciones técnicas de celulares utilizados
Fuente: Elaboración propia

Marca	Samsung
Modelo	Galaxy J5 Prime
Resolución camara trasera	CMOS de 13.0 MP

Tabla 4.3 Especificaciones técnicas de cámara IP utilizada
Fuente: Elaboración propia

Marca	EZVIZ
Modelo	CS-C8C

Ángulos PT	Pan: 352°, Tilt: 95°
Resolución	1920 × 1080
Compresión de video	H.265 / H.264
Standard Wi-Fi	IEEE802.11b, 802.11g, 802.11n
Red alámbrica	Puerto RJ45

Se hace uso de celulares para capturar las variables “Flujo de refrigerante – Rotámetro”, “Temperatura condensado – Probeta”, “Presión refrigerante - Estado 1” y “Presión refrigerante - Estado 3”. Cada variable tiene asignada un celular por lo que se cuenta con 4 celulares. Las especificaciones técnicas de los celulares se observan en la Tabla 4.2. Asimismo, se cuenta con una cámara IP de paneo con las especificaciones técnicas mostradas en la Tabla 4.3.

4.1.3 Controlador

El controlador estará conformado por un Arduino y un Raspberry, los cuales se comunican mediante un puerto serial, como se puede observar en la Figura 4.4.

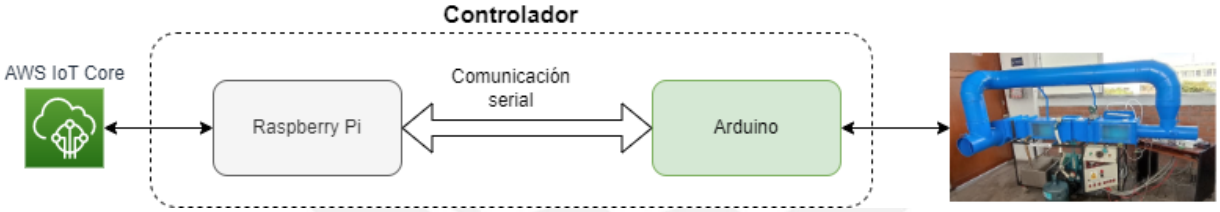


Figura 4.4 Componentes del controlador y comunicación entre ellas
Fuente: Elaboración propia

4.1.3.1 Configuración de Arduino

El Arduino utilizado tiene conectado los sensores a este. Los sensores son de tipo analógicas o digitales. Para los sensores de tipo digital se utiliza librerías DHT.h, para recibir mediciones del “Sensor de humedad relativa y temperatura DHT22”, y max6675.h para recibir mediciones del “transmisor MAX6675”. Para los sensores de tipo analógico, se utiliza la función de transferencia de cada sensor, la cual se puede obtener de los *datasheet* de los mismos. Las especificaciones técnicas del Arduino se puedes observar en la Tabla 4.4.

Tabla 4.4 Especificaciones del Arduino utilizado
Fuente: Elaboración propia

Modelo	Mega Rev3
Static RAM	8 KB
Microcontrolador	ATmega2560
Velocidad de clock	16 MHz
Cantidad de periféricos	1 puerto serial

4.1.3.2 Configuración de Raspberry

Debido a que el Arduino utilizado no puede utilizar el protocolo seguro por limitaciones de hardware, se propone el uso de un Raspberry para que sea este quien se conecte al AWS IoT Core, similar al despliegue desarrollado en [39], pero utilizando el protocolo MQTT y no WebSockets. Asimismo, se hace uso del Python SDK IoT Device proporcionado por AWS, el cual es una librería Python que permite a un dispositivo poder conectarse al IoT Core, enviar mensajes y recibir mensajes. Finalmente, se proporciona los certificados X.509 para identificar al dispositivo, descritos más adelante en 4.2.2. Las especificaciones técnicas del Raspberry se muestran en la Tabla 4.5.

Tabla 4.5 Especificaciones técnicas del Raspberry Pi
Fuente: Elaboración propia

Modelo	Raspberry Pi 3 Model B
Sistema operativo	Raspberry Pi OS
RAM	1 GB
Procesador	BCM2837

4.1.3.3 Comunicación entre Arduino y Raspberry

La comunicación entre el Arduino y el Raspberry se realizará mediante un puerto serial, como se muestra en la Figura 4.4. Para lo cual, el Arduino hace uso de su función nativa *Serial* y el Raspberry hace uso de *pyserial*, el cual es una librería que permite el acceso al puerto serial. El flujo de envío y recepción de *data* entre el Arduino y el Raspberry puede observarse en la Figura 4.5.

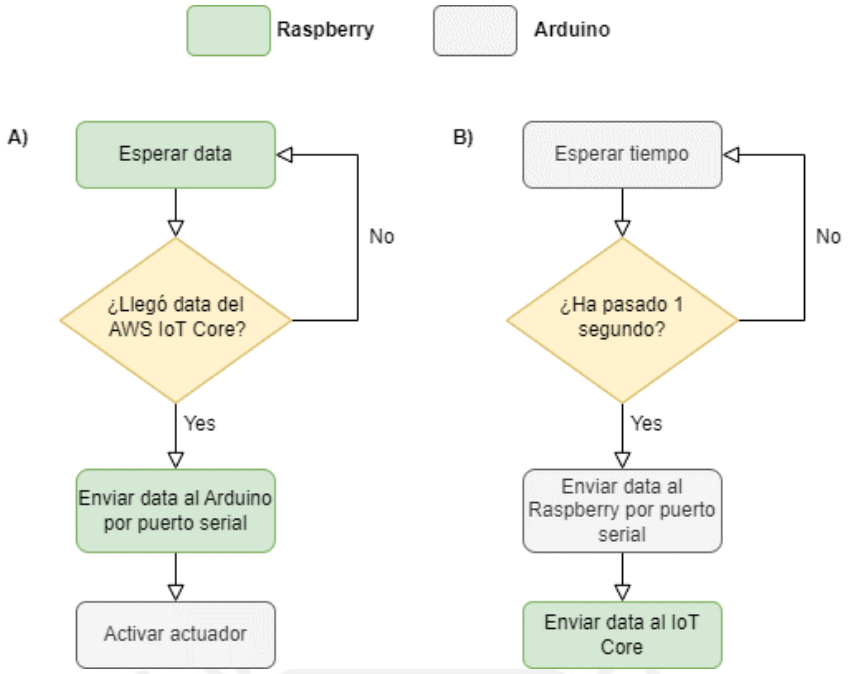


Figura 4.5 Flujo de envío y recepción de data entre Arduino y Raspberry
Fuente: Elaboración propia

4.2 Despliegue de AWS IoT Core

4.2.1 Creación de *things*

El Raspberry y el servidor Web estarán conectados al IoT Core. Así, para que dichos elementos puedan utilizar el IoT Core se debe crear una representación de cada uno de ellos en el servicio de AWS. A dicha representación AWS lo llama "thing", por lo que se crean dos *things*, uno que representa al EC2, donde se tendrá al Servidor Web, y otro que representa al banco de psicrometría, aunque será el Raspberry quien envié los mensajes.

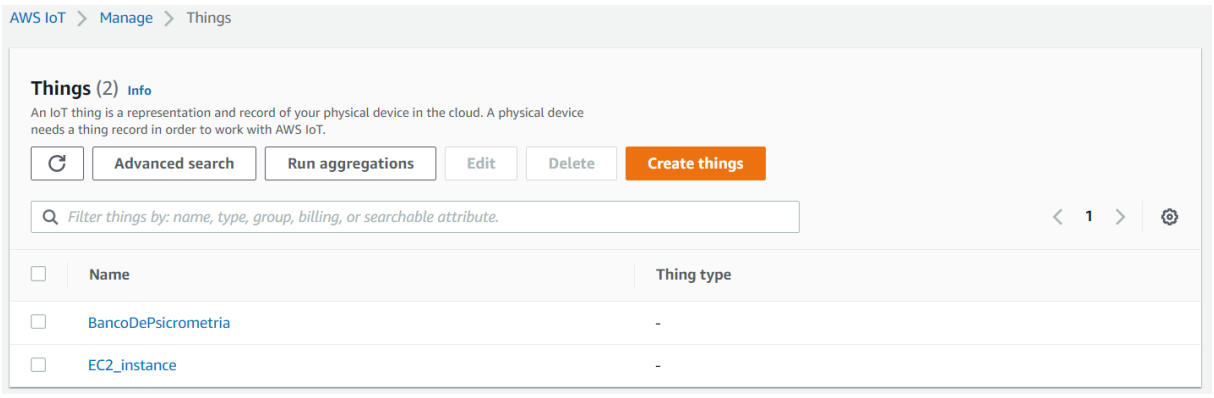


Figura 4.6 Things creados en la consola de AWS
Fuente: Elaboración propia

4.2.2 Configuración de seguridad

AWS IoT Core tiene dos niveles de seguridad: autenticación y autorización. La autenticación se realiza a través de certificados X.509, los cuales pueden ser generados al momento de crear un *thing*, realizado en el punto 4.2.1. Así, el Servidor Web y el Raspberry tienen su propio certificado X.509 que les permite identificarse con IoT Core. Por otro lado, la autorización se realiza a través de políticas de seguridad llamadas AWS IoT Core policy, las cuales son documentos JSON y determinan qué acciones pueden realizar un *thing* autenticado [49]. En el Anexo 2 se puede encontrar la política utilizada en donde se permite las acciones de conectarse al bróker AWS IoT Core de mensajes ("Action": "iot:Connect"), publicar a un topico MQTT ("Action": "iot:Publish"), suscribirse a un tópico filtro ("Action": "iot:Subscribe") y recibir mensajes ("Action": "iot:Receive"). Asimismo, no se especifica algún tipo de permiso en específico en cada una de las acciones antes mencionadas (por lo que todas tienen "Resource": "*").

4.2.3 Tópicos a utilizar y mensajes a enviar

Se utilizará el formato “LABEN2021/*tag_mqtt*” de tópico y el valor de la variable será en texto plano. Por ejemplo, se puede enviar el valor “19.6” al tópico LABEN2021/ TR_1, donde TR_1 es el tag_mqtt asignado al sensor de “Temperatura refrigerante - Estado 1”.

4.3 Despliegue EC2

4.3.1 Especificaciones técnicas

Las especificaciones técnicas de la instancia EC2 alquilada se puede observar en la Tabla 4.6. Los recursos listados son suficientes para una concurrencia de 8 usuarios al mismo tiempo, como se demostrará en el punto 4.5.3.

Tabla 4.6 Especificaciones técnicas de la instancia EC2
Fuente: elaboración propia

Tipo de instancia EC2	t2.micro
vCPU	1
Memoria RAM	1 GiB
Almacenamiento	16 GB
Sistema Operativo	Ubuntu 20.04.5 LTS

4.3.2 Servidor Web

Se instala la versión v16.17.0 de Node.js en la instancia ec2 alquilada y se implementa un servidor web utilizando el *framework* express.js.

4.3.2.1 Autenticación y autorización

Para cumplir el requerimiento R2 (acceso mediante credenciales o cuenta PUCP) y R3 (acceso automatizado basado en fecha y hora) se implementa el flujo de la Figura 4.7.

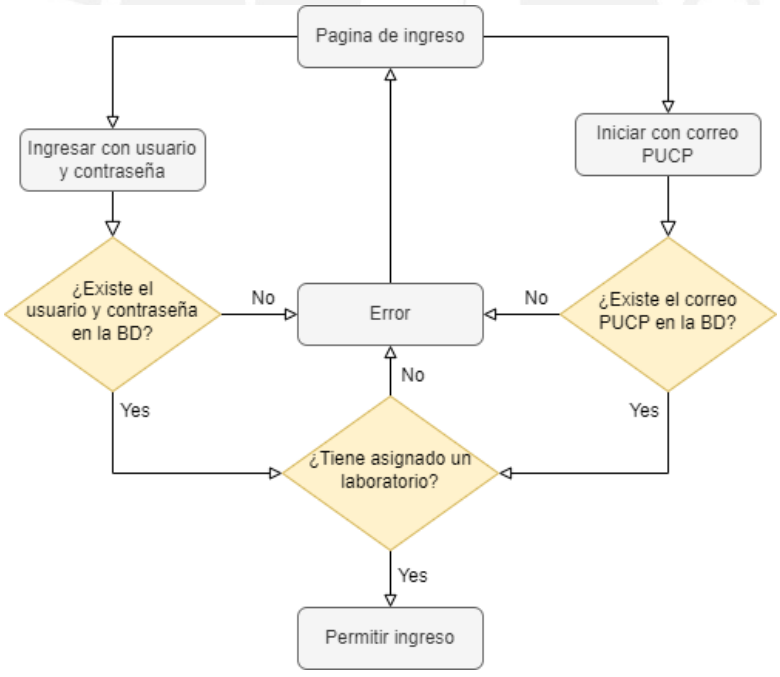


Figura 4.7 Flujo de autenticación y autorización a la plataforma
Fuente: Elaboración propia

La autenticación se realiza de dos maneras: mediante el protocolo OAuth 2.0, utilizando a Google como el *authorization server* (lo cual permite utilizar las cuentas PUCP), o mediante credenciales (usuario y contraseña). Para utilizar Google como el *authorization server* se debe

registrar la aplicación en la consola de Google Cloud y crear un “OAuth client ID”, el cual es utilizado para identificar a la aplicación Node.js a los servidores OAuth de Google.

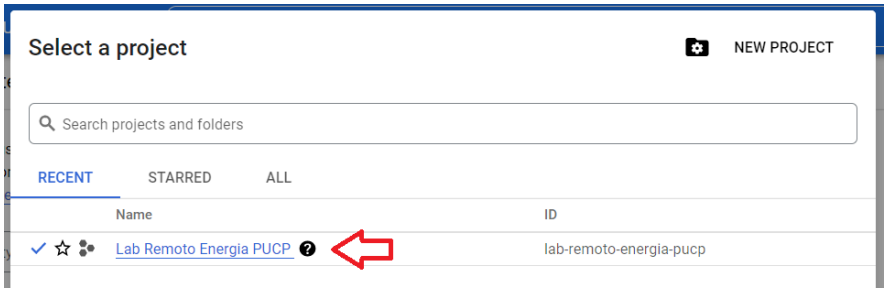


Figura 4.8 Proyecto creado en consola de Google Cloud
Fuente: Elaboración propia

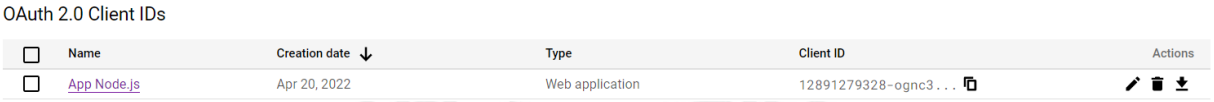


Figura 4.9 OAuth 2.0 Client ID creado en consola de Google Cloud para la aplicación Node.js implementada
Fuente: Elaboración propia

Para el uso del protocolo OAuth 2.0, se utiliza *Passport*, el cual es un *middleware* de autenticación para Node.js [50].

Para la autorización a la interfaz web del ensayo, se verifica si el usuario tiene asignado un ensayo al momento de ingreso. Si dicho usuario tiene asignado un ensayo, se permite el ingreso, sino se rechaza el ingreso.

En la Figura 4.10 se puede observar la vista de ingreso a la aplicación web, en donde se muestra que los dos métodos de ingreso: mediante credenciales o mediante cuenta institucional PUCP.

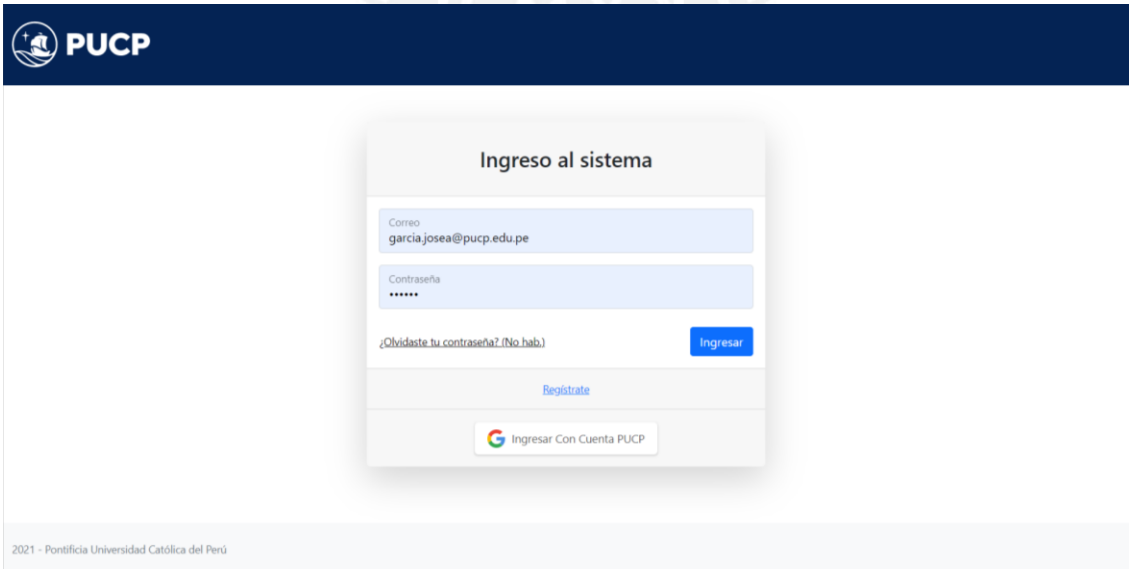


Figura 4.10 Vista de ingreso a la aplicación web
Fuente: elaboración propia

4.3.2.2 Implementación de Web Sockets

Para cumplir los requerimientos R5 (asignación de interruptores a controlar en tiempo real), R6 (visualización de alumnos conectados en tiempo real), R7 (visualización de data en tiempo real) y R8 (control del equipo en tiempo real) se utilizará WebSockets y, en particular, se agrupa las conexiones bidireccionales de los usuarios activos en administradores, jefes de practica y alumnos. Así, se podrá enviar un mensaje a todos los usuarios, solo a los alumnos, solo a los jefes de prácticas, solo a los administrados o cualquier combinatoria entre los tres últimos mencionados (mensajes solo para los administradores y jefes de práctica, pero no a los alumnos, por ejemplo).

Para el cumplimiento del requerimiento R5, se realiza el flujo observado en la Figura 4.11. Se puede observar que una vez que un administrador o jefe de práctica asigna una o varias variables a algún estudiante, se envía al servidor el id del estudiante y el de las variables a ser asignadas. Luego el servidor envía dichos id al resto de administradores y jefes de prácticas para evitar que estos asignen dichas variables a otros estudiantes al mismo tiempo. Asimismo, se envía el id de las variables al estudiante al cual se le ha sido asignada para que su interfaz se actualice y pueda visualizar el interruptor o los interruptores que se le han sido asignado.

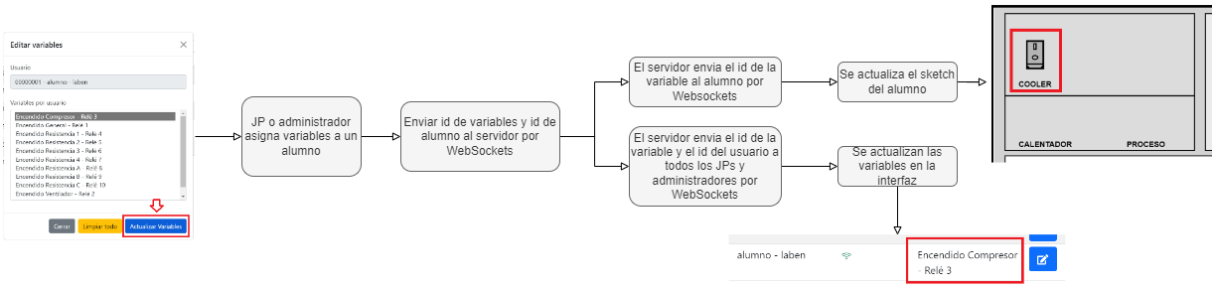


Figura 4.11 Flujo de asignación de variables
Fuente: Elaboración propia

Para el cumplimiento del requerimiento R6, se realiza el flujo observado en la Figura 4.12. Se puede observar que una vez que se detecte si existe una conexión o desconexión de un alumno, se envía el id del alumno a todos los jefes de práctica y administradores para que se actualice su estado en tiempo real.

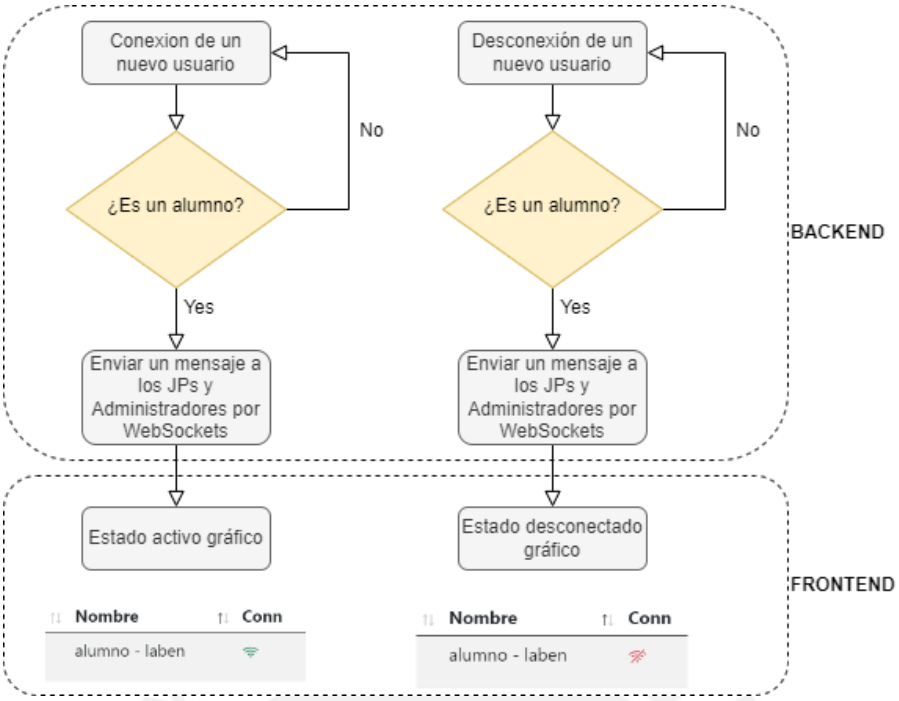


Figura 4.12 Flujo de visualización de conexión y desconexión de alumno
Fuente: Elaboración propia

Para el cumplimiento del requerimiento R7, se realiza el flujo observado en la Figura 4.13. Se puede observar el flujo completo de la *data* para un mejor entendimiento. Una vez que la *data* llega al Servidor Web, este reenvía dichos valores a todos los usuarios conectados (alumnos, jefes de prácticas y administradores) para mostrárselos en sus interfaces web. Asimismo, una vez que la data llega a la interfaz del usuario, se puede agregar dicho valor a un gráfico "tiempo vs. valor" para ver los nuevos valores en tiempo real.

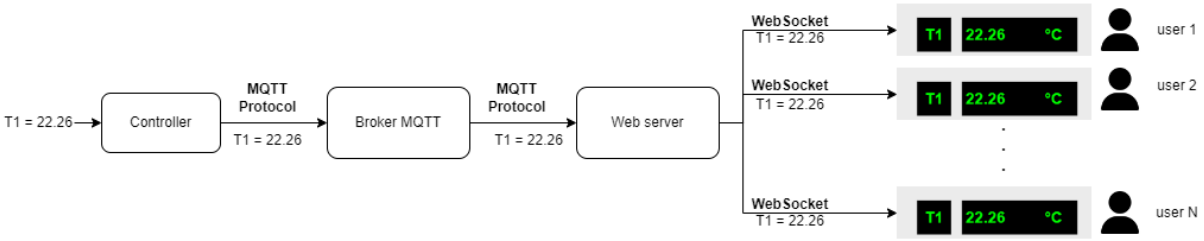


Figura 4.13 Flujo de visualización de data en tiempo real
Fuente: Elaboración propia

Para el cumplimiento del requerimiento R8, se realiza el flujo observado en la Figura 4.14. Se puede observar el flujo completo de la *data* para un mejor entendimiento. Una vez que algún usuario encienda o apague algún interruptor, se envía el nuevo valor al servidor a través de WebSockets. El servidor realiza dos acciones al recibir dicho valor. En primer lugar, reenvía

dichos valores a todos los usuarios conectados (alumnos, jefes de prácticas y administradores) con la finalidad de que estos puedan observar la acción realizada por el estudiante en tiempo real. En segundo lugar, publica el nuevo valor al IoT Core para que el nuevo valor llegue al controlador, quien activará el actuador correspondiente. Así, el estudiante logra controlar el equipo remotamente.

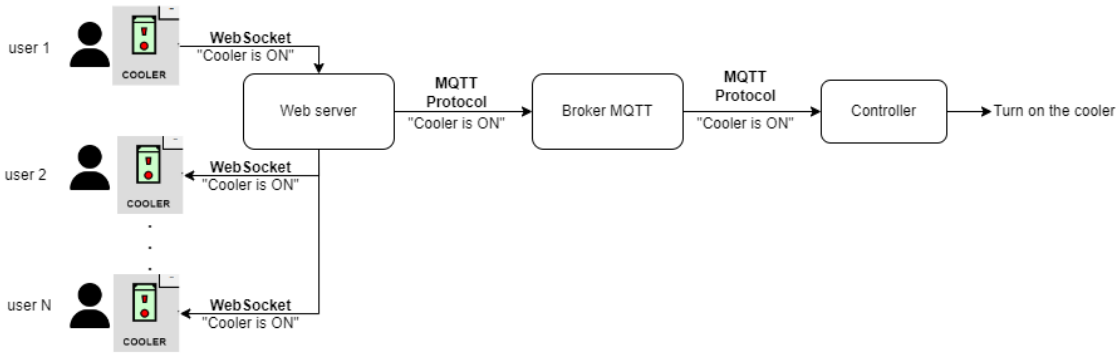


Figura 4.14 Flujo de control remoto del banco de psicrometría
Fuente: Elaboración propia

4.3.2.3 Visualización de variables de video

Para visualizar las variables que no cuentan con un sensor (requerimiento R9) se usará una sesión *meet* por el cual el estuante podrá visualizar la variable; por ejemplo, el nanómetro. En la interfaz web se encontrará botones los cuales al ser presionados redireccionaran a la sesión *meet*. El enlace de dichas variables se almacena en la tabla “datos_meeting_link” de la base de datos, la cual se mostró en la Figura 3.2.

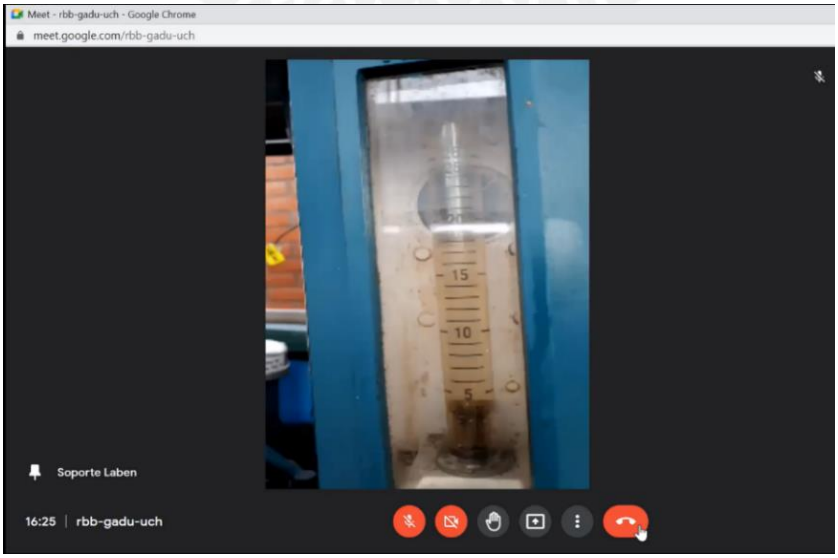


Figura 4.15 Video de la variable “Flujo de refrigerante – Rotámetro”
Fuente: Elaboración propia

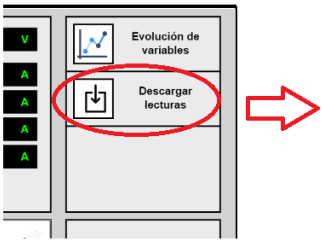
4.3.2.4 Interfaz de usuario

Para la implementación del interfaz de usuario, se reutilizó la utilizada en los simuladores mencionados en el punto 1.2, la cual fue implementada por personal del Laboratorio de Energía. Los simuladores utilizaban la librería p5.js, la cual tiene funcionalidades para realizar gráficos. Así, se logra tener una "versión de interfaz web" del equipo físico, como puede observarse en la Figura 4.16. Se adaptó el código utilizado para ser usado por el Servidor Web de tal manera que las acciones realizadas en la interfaz se envíen a través de WebSockets al Servidor y este reenvíe la acción al Bróker IoT Core para que, finalmente, el equipo físico realice la acción solicitada.

Figura 4.16 Equipo físico (izquierda) y versión de interfaz (derecha)
Fuente: Elaboración propia

4.3.2.5 Descarga de *data* del ensayo

Para cumplir el requerimiento R11 (exportar resultados a csv), hace uso de la librería *Fast-csv*, la cual permite formatear archivos CSV en Node.js [51].



	A	B	C	D
1	fecha	Corriente eléctrica - Resistencia 1	Corriente eléctrica - Resistencia 2	Corriente eléctrica - Resistencia 3
2	15/09/2022 8:29:07	0.00	0.00	0.00
3	15/09/2022 8:29:08	0.43	0.40	0.41
4	15/09/2022 8:29:09	0.43	0.41	0.40
5	15/09/2022 8:29:10	0.41	0.39	0.41
6	15/09/2022 8:29:11	0.42	0.40	0.41
7	15/09/2022 8:29:12	0.41	0.40	0.41
8	15/09/2022 8:29:13	0.41	0.41	0.41
9	15/09/2022 8:29:14	0.41	0.41	0.40
10	15/09/2022 8:29:15	0.44	0.40	0.41
11	15/09/2022 8:29:16	0.41	0.40	0.41

Figura 4.17 Ejemplo archivo CSV descargado
Fuente: Elaboración propia

4.3.3 Gestor de aplicaciones Node.js

Una vez finalizado el desarrollo de la aplicación Node.js, se hace uso de PM2 [52] para que la aplicación corra en segundo plano como un servicio.

4.4 Despliegue del RDS

4.4.1 Especificaciones técnicas

Las especificaciones técnicas del RDS alquilado se puede observar en la Tabla 4.7. Los recursos listados son suficientes para la escritura de *data*, como se verá más adelante en 4.5.4 más adelante, y el almacenamiento de la *data* enviada por el equipo en un ciclo académico, como se validó en 4.8.

Tabla 4.7 Especificaciones técnicas del RDS
Fuente: elaboración propia

Tipo de instancia	db.t3.micro
vCPU	2
Memoria RAM	1 GiB
Almacenamiento	20 GB

4.5 Pruebas de rendimiento

Se realizarán dos pruebas de rendimiento: al Servidor Web, con la finalidad de cumplir el requerimiento R10 (concurrencia de 8 usuarios), dentro de la instancia EC2 y a la base de datos en la RDS. Las pruebas serán realizadas con la finalidad de comprobar si las especificaciones técnicas configuradas en un primer momento de la instancia EC2 y de la instancia RDS serán suficientes o necesitarán incrementarse. AWS permite un fácil escalamiento, por lo que incrementar o reducir recursos se realiza de manera rápida y sencilla. Así, para realizar las pruebas al Servidor Web y la RDS se utilizará Apache JMeter y un archivo Python, respectivamente.

4.5.1 Archivo Python

Se realiza un programa en Python que simule el envío de mensajes de los sensores, el cual será ejecutado en el Raspberry Pi. El archivo Python puede encontrarse en el Anexo 3.

4.5.2 Configuración del JMeter

JMeter permitirá simular los usuarios que utilizarán el Servidor Web. Para realizar las simulaciones de carga se utilizará los elementos mostrados en la Figura 4.18 dentro del "test plan". Se puede observar que se utilizará un "Thread Group", un "HTTP(S) Test Script Recorder" y un "Recording Controller".

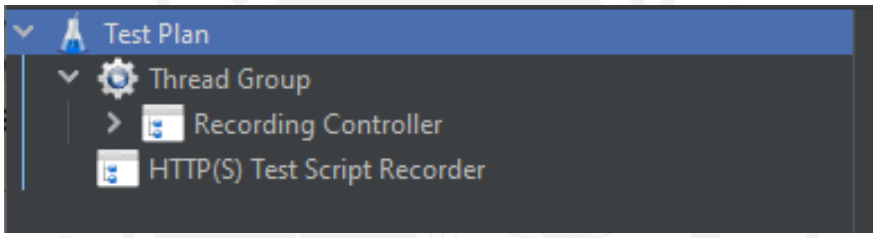


Figura 4.18 Configuración del "Test Plan" en JMeter
Fuente: Elaboración propia

En primer lugar, se configura el "Thread Group" como se observa en la Figura 4.19. En dicha figura se observa las configuraciones realizadas, las cuales son las siguientes:

- Number of Threads: Se configura con un valor de 10, pues es la cantidad de usuarios a simular. Se considera un valor mayor a la cantidad de usuarios que usualmente utilizarán la plataforma al mismo tiempo, el cual es de 6 en promedio, para tener un margen superior.
- Ramp-up period: Se configura con un valor de 0, pues se requiere una concurrencia de todos los usuarios al mismo tiempo. Se considera el peor de los casos, donde todos realizan las mismas acciones al mismo tiempo.
- Loop Count: Se configura con un valor de 5, pues se desea realizará 5 simulaciones.

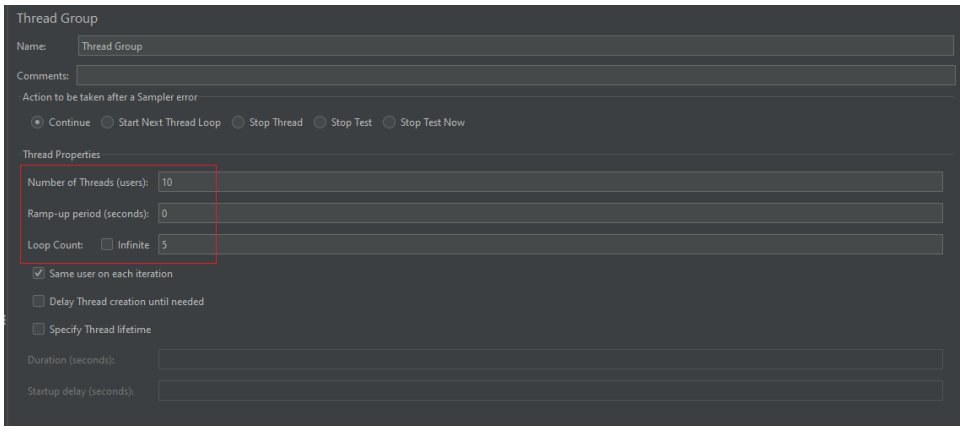


Figura 4.19 Configuración del " Thread Group " en JMeter
Fuente: Elaboración propia

En segundo lugar, se configura el elemento "HTTP(S) Test Script Recorder" para facilitar las simulaciones. Se dejará las configuraciones por *default*. Al arrancar este elemento se levantará un servidor *proxy* en el puerto 8888.

Finalmente, se utilizará el navegador web Firefox para realizar las acciones que se desea simular. Así, es necesario configurar el servidor *proxy* como se observa en la Figura 4.20.

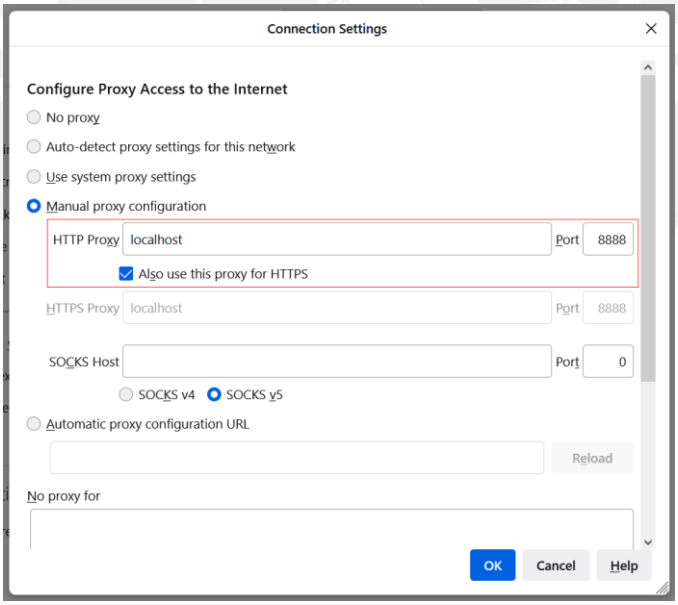


Figura 4.20 Configuración de servidor proxy en navegador web Firefox
Fuente: Elaboración propia

Las configuraciones realizadas serán las mismas para cada prueba, pero las solicitudes HTTPS almacenadas en el "Recording Controller" dependerá del flujo a ser realizado. Los pasos a seguir son los siguientes:

- Se activa el "HTTP(S) Test Script Recorder".

- Se realiza el flujo a ser simulado en Firefox para almacenar las acciones realizadas en el “Recording Controller” y pueden ser utilizadas en la simulación.
- Se guarda el *test* y se corre el comando “jmeter -n -t *test_name*.jmk”, donde *test_name* es el nombre del *test* a correr por JMeter.

4.5.3 Prueba de *performance* de instancia EC2

En la instancia EC2 se encontrará la aplicación web implementada. Según los requerimientos listados en 3.1, es necesario que exista aproximadamente 8 usuarios en concurrencia. Para lo cual, se realizará dicha prueba de rendimiento. En particular, se evaluará el porcentaje de CPU y la memoria utilizados por la aplicación Node.js implementada. Se utilizará JMeter para simular la concurrencia de 10 usuarios para tener un margen superior al requerimiento utilizando y se realizará 5 simulaciones para capturar el mayor pico que se obtenga y será ese instante el que se mostrará. Se realizará la prueba para dos flujos: el flujo de gráfico de *dashboards* y la descarga de evolución de variables, ya que son dichas acciones las que más recursos consumirán, debido a la cantidad de datos a manejar.

Para lograr monitorear los parámetros anteriormente mencionados, se utilizará una funcionalidad de PM2 llamada *pm2 monitor* la cual brinda *dashboards* en tiempo real de los recursos consumidos por la aplicación Node.js.

4.5.3.1 Prueba de rendimiento en gráfico de evolución de variables

Se capturo el flujo desde que se ingresa a la plataforma hasta que se grafican las variables por el servidor *proxy*, se gráfica 4 variables debido a que esa es la cantidad promedio de variables a graficar simultáneamente para realizar comparación, como la evolución de las 4 resistencias (Corriente eléctrica - Resistencia 1, 2, 3 y 4), por ejemplo.

En la Figura 4.21 se puede observar el porcentaje de CPU y memoria consumida por la aplicación en la prueba realizada. Se puede notar que dichos valores son bajos y, por ende, el

servidor muestra un buen rendimiento, pues el porcentaje de CPU no supera el 20% y la memoria consumida solo es de aproximadamente 119MiB, lo cual equivale solo al 12% aproximadamente de la memoria RAM configurada (1 GiB).

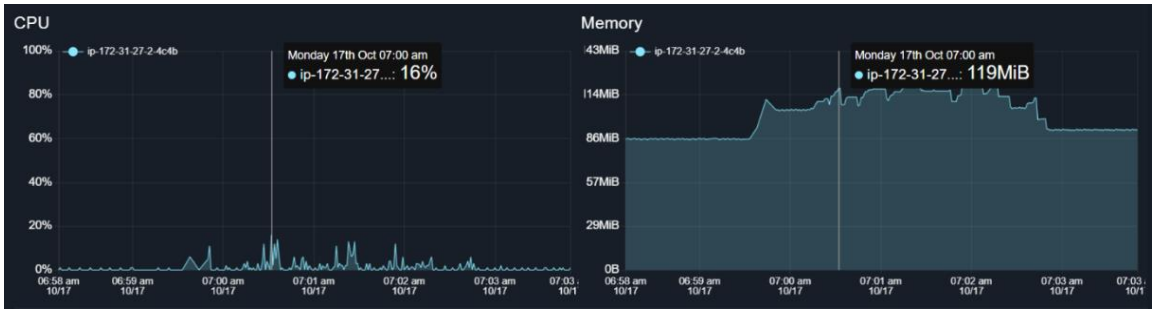


Figura 4.21 CPU y memoria utilizada en prueba de rendimiento de gráfico de evolución de variables
Fuente: elaboración propia

4.5.3.2 Prueba de rendimiento en descarga de archivo CSV.

Se capturo el flujo desde que se ingresa a la aplicación web hasta que se presiona el botón de descarga de datos.

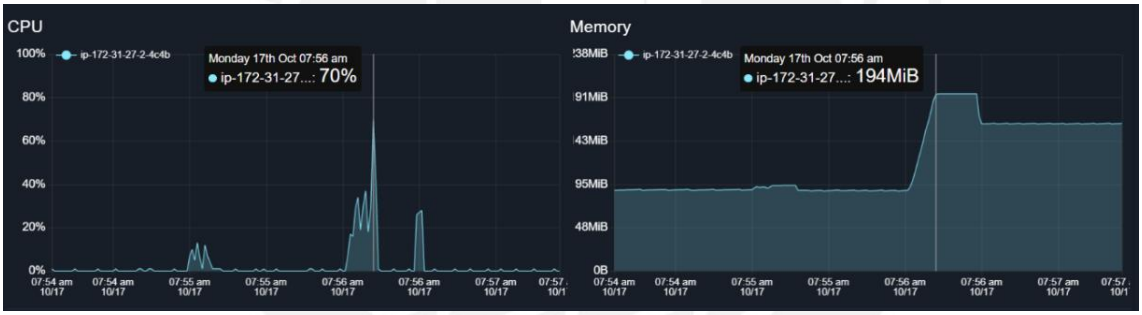


Figura 4.22 CPU y memoria utilizada en prueba de rendimiento de descarga de archivo CSV
Fuente: elaboración propia

En la Figura 4.22 se puede observar el porcentaje de CPU y memoria consumida por la aplicación en la prueba realizada. Se puede notar que ambos valores aumentaron, pero el CPU es notablemente mayor al caso anterior, ya que se tiene un pico de 70%. Es comprensible, debido a que se descarga la data de todas las variables de entrada, las 22, y el *query* ejecutado para obtener dicha *data* es complejo. Sin embargo, se considera aceptable, ya que no alcanza picos mayores al 90% y que se utiliza el *query* complejo debido a que es requisito brindado por parte del personal del laboratorio de descargar los datos recolectados de todas las variables en conjunto.

4.5.4 Prueba de performance RDS

Se hace uso del *dashboard* de rendimiento generado por MySQL Workbench y los resultados se pueden observar en la Figura 4.23. Se puede observar que se ejecutan aproximadamente 22 *querys* INSERT, lo cual corresponde debido a las 22 variables de entrada de la Tabla 3.3.

Figura 4.23 Dashboard *de rendimiento generado en la prueba realizada*
Fuente: elaboración propia

Se puede notar que se mantiene una eficiencia del 99% en "Table Open Cable" lo que asegura que *data* solicitada varias veces de una misma tabla será brindada lo más rápido posible, pues la tabla se mantiene "abierta". Asimismo, se puede notar que el tamaño de la *data* escrita en disco es relativamente bajo, ya que está en el orden de los KB/s, lo cual puede observarse del parámetro "Inno DB Disk Writes".

4.6 Ensayos reales realizados con estudiantes

En el semestre 2022-1, el Laboratorio de Energía coordinó la realización de laboratorios completamente remoto utilizando la plataforma desarrollada con la finalidad de probar su uso en un entorno real y, además, obtener *feedback* de los estudiantes del uso de la misma. Asimismo, sería útil para saber si algún punto no fue cubierto con las pruebas realizadas en los últimos dos puntos.

El laboratorio fue realizado tal cual se realizaba con el uso de los simuladores: los estudiantes se reunían en una reunión zoom, el JP explicaba de qué trataba el laboratorio (los fundamentos teóricos, las experiencias y análisis a realizar, etc) y, luego, los alumnos realizaban simulaciones para completar el reporte de su laboratorio. Sin embargo, en vez de los simuladores, se utilizó la plataforma implementada. El JP explicó la dinámica del laboratorio, cómo utilizar la plataforma y asignó las variables a ser controlada por diferentes alumnos. En las Figura 4.24 y Figura 4.25 se puede observar la sesión zoom del laboratorio realizado el 14 de junio del 2022, semestre académico 2022-1, donde se observa al JP explicando el uso de la plataforma.

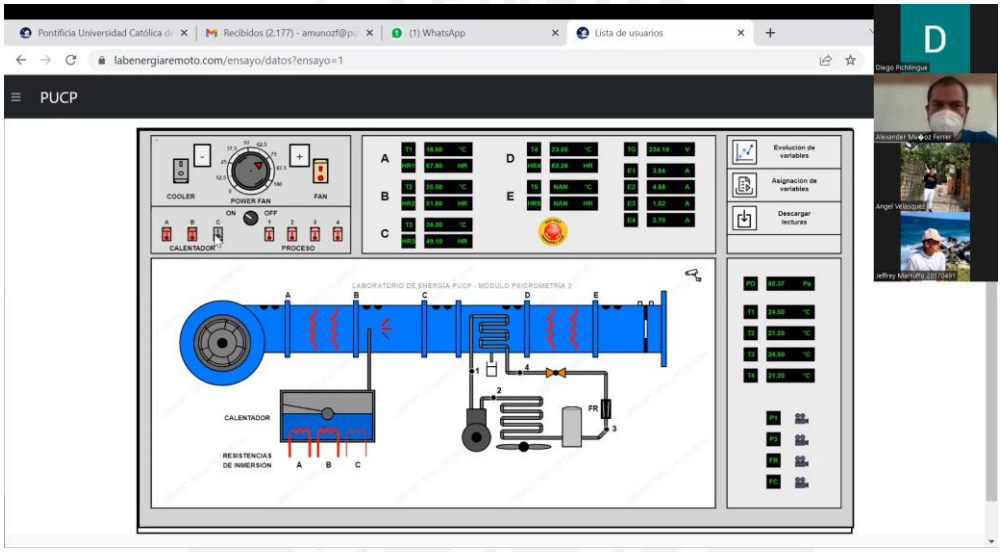


Figura 4.24 JP explicando el uso de la interfaz web
Fuente: grabación zoom de la reunión realizada

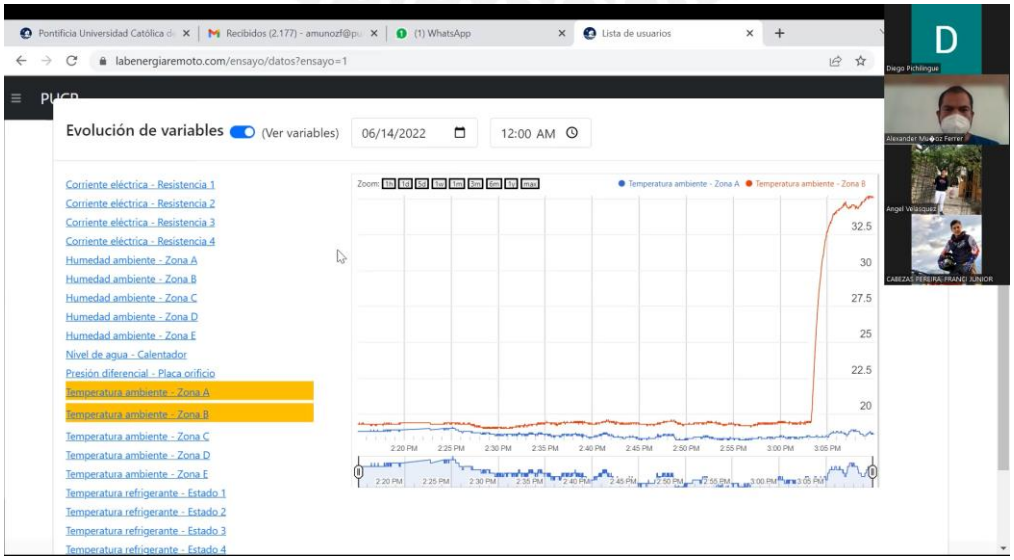


Figura 4.25 JP explicando la gráfica de evolución de variables
Fuente: grabación zoom de la reunión realizada

4.6.1 Resultado de encuestas

Después de los laboratorios realizados mencionados en el punto anterior se realizaron encuestas a los estudiantes participantes, de los cuales se pudo detectar dos puntos de mejora: “lentitud” al momento de graficar la evolución de las variables y *delay* existente en la vista del ensayo.

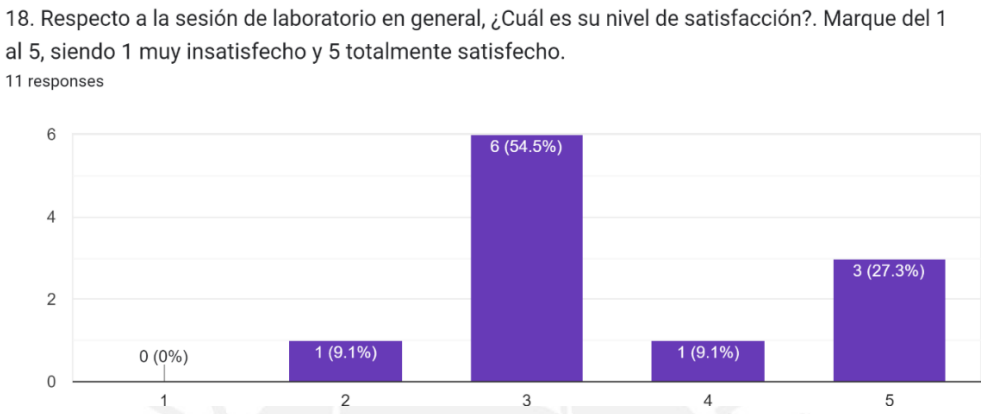


Figura 4.26 Nivel de satisfacción de los estudiantes obtenido
Fuente: formulario realizado por personal del Laboratorio de Energía de la PUCP

4.6.1.1 Detección de puntos de mejora

En primer lugar, se tiene la lentitud expresada por los estudiantes. Esta se debía a la librería utilizada para realizar los gráficos, Google Charts, pues presentaba dos inconvenientes. Por un lado, se necesitaba realizar una consulta a base de datos de todas las variables a graficar en conjunto. Es decir, para graficar una variable, se obtenía la *data* solo de dicha variable. Sin embargo, para graficar una variable adicional, se obtenía la *data* de ambas variables. En otras palabras, se volvía a obtener *data* de las variables anteriormente graficadas en lugar de solo obtener *data* de aquellas extras que se desee graficar. Así, el *delay* en obtener toda la *data*, a medida que aumentaban la cantidad de variables a graficar, aumentaba. Por otro lado, una vez la *data* obtenida de base de datos llegaba al usuario final, la librería presentaba ineficiencia al mostrar cantidades grandes de datos en el *dashboard*, lo que generaba la sensación de “lentitud” del estudiante.

En segundo lugar, se identificó un problema al ingresar a la vista del ensayo. Se detectó que se demoraba en terminar de renderizar el gráfico del equipo. Esto era debido a que se necesita

obtener los últimos valores almacenados en base de datos de las variables de entrada para mostrárselos a los usuarios. Para obtener dichos valores se ejecutaba un *query* complejo en la tabla *valor_variables*, pues se realizaba dos *subquerys* en la tabla *valor_variables*, el cual es la tabla que tiene mayor cantidad de *data* almacenada, ya que en dicha tabla se almacena la evolución que tiene cada variable del ensayo. Así, la ejecución de dicho *query* tomaba un tiempo en el orden de los segundos, el cual era perceptible por el usuario y generaba la sensación de "lentitud" del estudiante.

Ambos puntos causaron que el nivel de satisfacción de los estudiantes tenga una tendencia mayor a ser "intermedio" (representado con el numero 3), como puede observarse en la Figura 4.26, y no a ser "satisfecho" o "totalmente satisfecho", que es lo buscado.

4.6.2 Mejoras realizadas

En primer lugar, para solucionar el problema de generación de *dashboard*, se buscó una librería optimizada para generar un *dashboard* con una gran cantidad de data, por lo que se optó por librerías de renderizado para *big data*. Finalmente, se optó por la librería llamada Highcharts que brinda un *dashboard* capaz de graficar medio millón de puntos, lo cual es suficiente, ya que la data generada por laboratorio es aproximadamente de 300 000 si se grafica todas las variables de entrada (por segundo se genera 21 valores y el laboratorio dura 4 horas). Asimismo, las series de *data* a gráficar se pueden agregar de manera "aditiva". Es decir, para agregar una serie de datos solo es necesario agregar dicha serie al objeto representativo de la *data* a graficar, lo que genera que no se tenga que volver a solicitar toda la data a la base de datos cada vez que se desea agregar una serie más al *dashboard*. Así, por cada serie a graficar se realiza un simple *query* SQL de SELECT, caso contrario a los señalado en el punto anterior.

En segundo lugar, para solucionar la demora al renderizar la gráfica del equipo, se optó por almacenar el valor en dos tablas cada que el valor llega a la plataforma. Se inserta el nuevo

valor en la tabla *valor_variables*, para almacenar la evolución de valores, y se realiza una actualización al atributo *ultimo_valor* añadido a la tabla *variables_ensayo*. Así, para renderizar la pestaña se realiza un SELECT simple a la tabla *variables_ensayo* y se utiliza los valores del atributo *ultimo_valor*.

4.6.3 Resultado de encuestas con mejoras realizadas

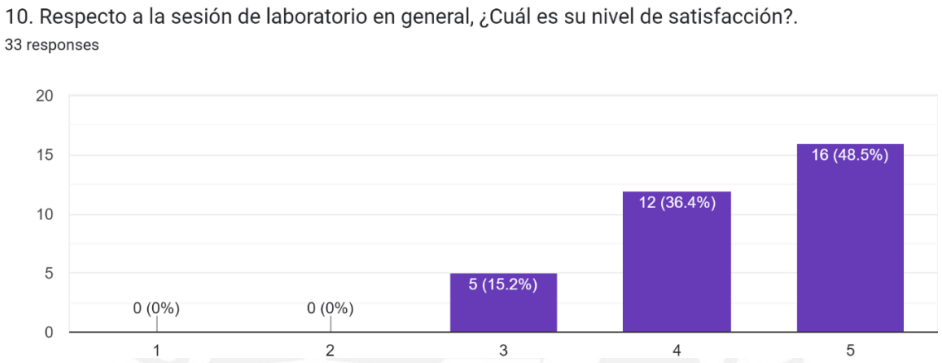


Figura 4.27 Nivel de satisfacción de los estudiantes después de implementar mejoras
Fuente: Formulario realizado por personal del Laboratorio de Energía de la PUCP

Como puede observarse en la Figura 4.27, con las mejoras implementadas mencionadas en el punto anterior se obtuvo un nivel que la tendencia del nivel de satisfacción de los estudiantes fue ser “satisfecho” o “totalmente satisfecho”, que fue lo buscado.

4.7 Configuraciones finales para producción

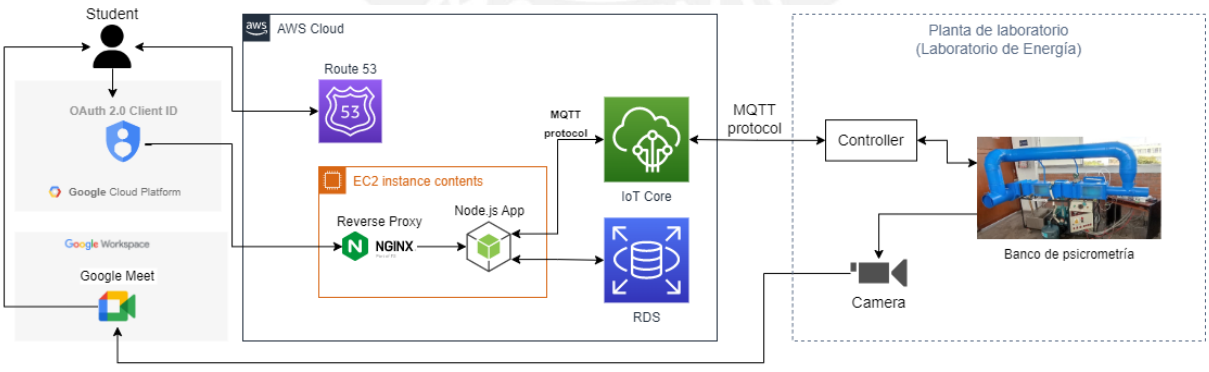


Figura 4.28 Arquitectura final de la plataforma
Fuente: elaboración propia

Se agrega dos componentes finales a la arquitectura propuesta. Los componentes agregados son el servidor Ngnix como *proxy* inverso dentro de la instancia EC2 y el servicio “Route 53” de AWS, como puede observarse en la Figura 4.28.

4.7.1 Amazon Route 53

Amazon Route 53 es un servicio web de DNS (*Domain Name System*) escalable y de alta disponibilidad [53]. Se utiliza dos funcionalidades del servicio: registro de nombre de dominio y ruteo DNS.

En primer lugar, se realizó el registro el nombre de dominio *labenergiaremoto.com* con la finalidad de que los alumnos accedan fácilmente al Servidor Web, ya que es más sencillo recordar *labenergiaremoto.com* que la IP pública de un servidor.

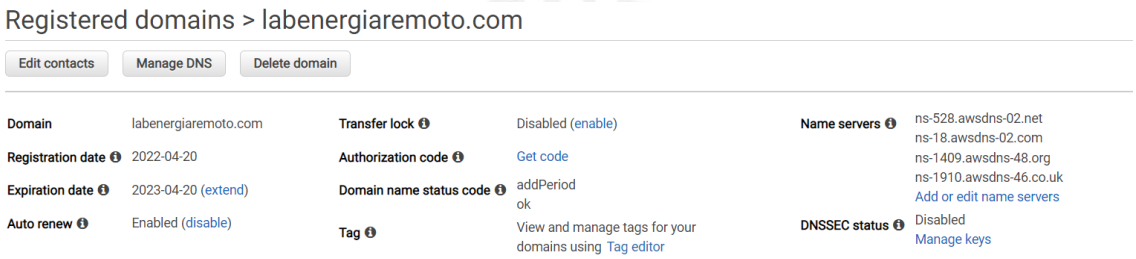


Figura 4.29 Dominio registrado en Route 53
Fuente: elaboración propia

En segundo lugar, Route 53 permite rutear tráfico de internet hacia la aplicación web implementada. Para lograrlo, se tiene los *hosted zones* públicas. Una *hosted zone* publica es un contenedor de reglas que definen cómo se rutea el tráfico en internet para un dominio especifico o sus subdominios [54]. Así, se crea una *hosted zone* pública y se configura que el tráfico dirigido a *www.labenergiaremoto.com* se rutee a la IP pública de la instancia EC2, como se observa en la Figura 4.30.

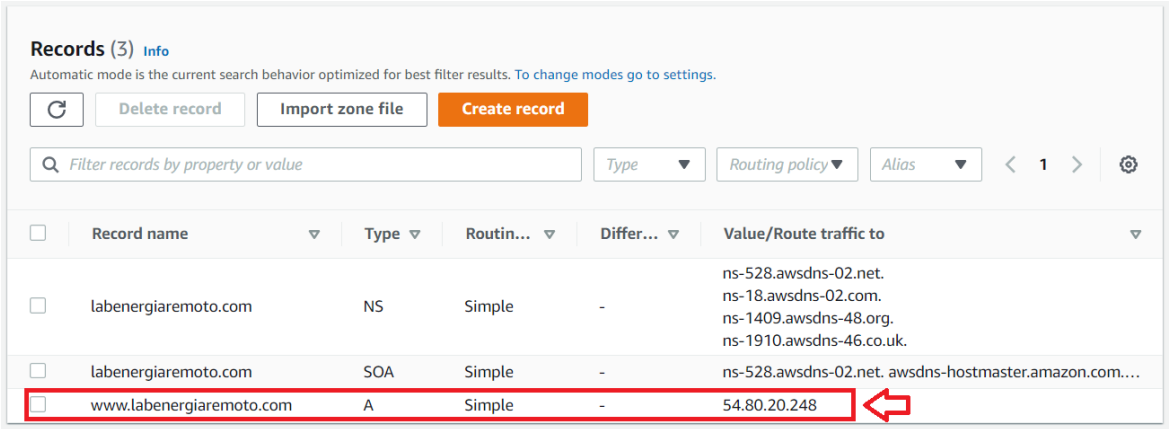


Figura 4.30 Configuración de redirección del tráfico al nombre de dominio a la instancia EC2
Fuente: elaboración propia

4.7.2 Servidor NGINX como *proxy* inverso

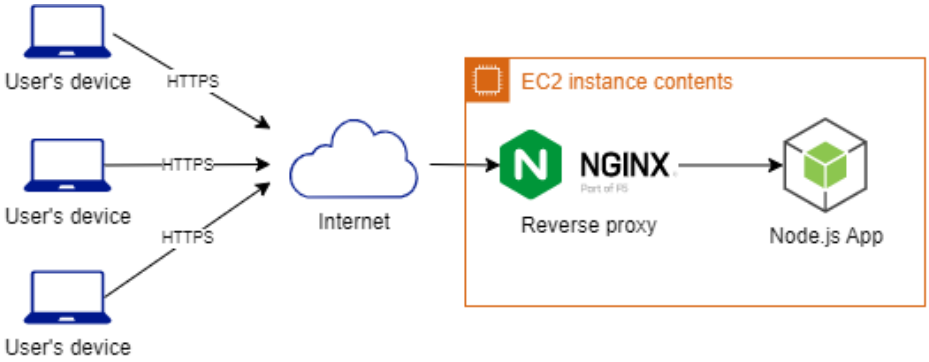


Figura 4.31 Flujo del proxy inverso
Fuente: elaboración propia

NGINX es un servidor HTTP y *proxy* inverso gratis, de código abierto y de rendimiento alto. Es conocido por su rendimiento alto, estabilidad, conjunto de *features*, configuración simple y bajo consumo de recursos [55]. Así, se instala el NGINX en la instancia EC2 y se obtiene un certificado SSL para NGINX brindado por *Let's Encrypt*, el cual es una autoridad de certificado (CA) [56], para habilitar el protocolo HTTPS en el servidor NGINX. Asimismo, se usa Certbot. Certbot es una herramienta de *software* gratuito y de código abierto para automatizar el uso de certificados *Let's Encrypt* en sitios web administrados manualmente para habilitar HTTPS [57]. De esta manera, la renovación del certificado SSL se realizará de forma automática y no necesitará ser realizado de forma manual. Así, una vez configurado el servidor NGINX con su certificado SSL, se puede verificar su correcto funcionamiento a través del SSL Server Test, como se observa en la Figura 4.32.

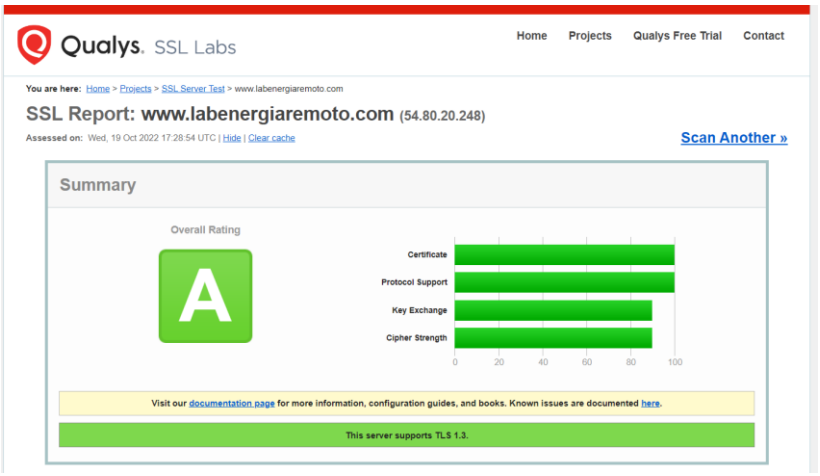


Figura 4.32 Reporte generado por SSL Server Test para www.labenergiaremoto.com
Fuente: elaboración propia

Una vez se puede utilizar HTTPS, se configura NGINX como *proxy* inverso. Así, será el encargado de interceptar las solicitudes de los clientes y reenviarlos a la aplicación Node.js, como se observa en la Figura 4.31. El implementar un *proxy* inverso nos brinda beneficios como el ser un balanceador de carga, lo que permitirá tener más servidores web para una cantidad mayor de laboratorios a ser implementados a un futuro.

4.8 Estado actual de la plataforma

En la actualidad, a fecha de escrita esta tesis (octubre del 2022), la plataforma está siendo utilizada en laboratorios presenciales. A pesar de que los laboratorios ya no son remotos, debido a las nuevas disposiciones del Gobierno Peruano, el Laboratorio de Energía ha optado por utilizar la plataforma en manera conjunta con el laboratorio presencial con dos finalidades: seguir obteniendo *feedback* de los alumnos para una mejora continua de la plataforma y que estos conozcan la plataforma y se apropien de la solución. La plataforma está siendo utilizada a lo largo de todo el semestre 2022-2, sin presentar inconveniente alguno, para controlar el equipo utilizando computadoras locales colocadas por el personal del laboratorio, como puede verse en Figura 4.34, o los alumnos pueden llevar sus propios laptops.

Figura 4.33 JP explicando el desarrollo de un laboratorio a los estudiantes
Fuente: imagen obtenida de la cámara de paneo del Laboratorio de Energía

Figura 4.34 Estudiantes utilizando la plataforma con computadoras locales
Fuente: imagen obtenida de la cámara de paneo del Laboratorio de Energía

Asimismo, los alumnos ya no necesitan medir constantemente el estado del aire dentro del equipo con medidores convencionales, sino que pueden analizar cómo va variando dicho estado generando *dashboards* en tiempo real. En la Figura 4.35 puede observarse la evolución de la temperatura del aire en las zonas A, B, C, D y E de un laboratorio realizado el 15 de septiembre del 2022.

Figura 4.35 Temperaturas ambiente de todas las zonas - laboratorio realizado el 15 de septiembre del 2022
Fuente: Elaboración propia

Finalmente, se puede analizar cuánto almacenamiento ha sido utilizado hasta el momento y analizar si el almacenamiento configurado es suficiente. Se puede obtener el espacio de almacenamiento libre desde la consola de AWS de la base de datos y se puede realizar un

cálculo sencillo con el desarrollo de un laboratorio realizado el 25/10/2022. En la Figura 4.36 se puede observar que se tiene 18.381 GB de almacenamiento libre antes del laboratorio y 18.345 GB después del mismo (Figura 4.37). Así, se consumió 35.98 MB en 3.3 horas, que fue el tiempo total de envió de datos del ensayo, lo que da un aproximado de 43.62 MB, por regla de tres simple, en 4 horas. Como se verá en el punto 5.2.2, la cantidad promedio de laboratorios por semestre académico es de 51 lo que resulta en 2.22 GB (suficiente con los 20GB configurado).

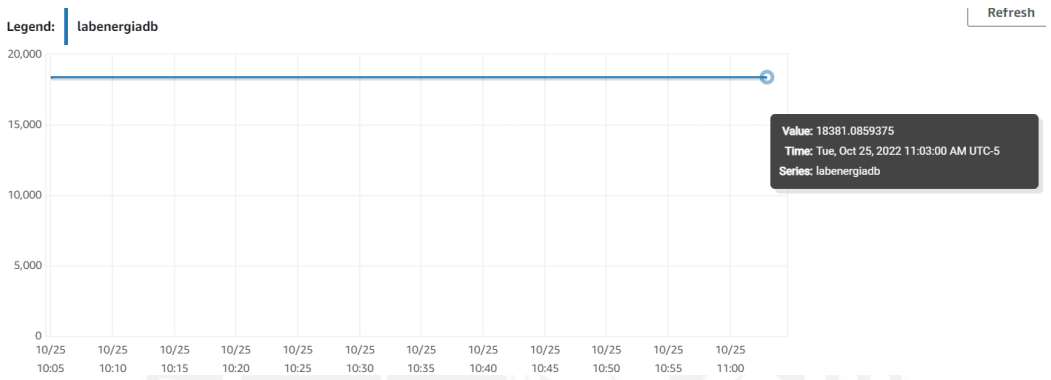


Figura 4.36 Espacio de almacenamiento libre antes del laboratorio
Fuente: elaboración propia

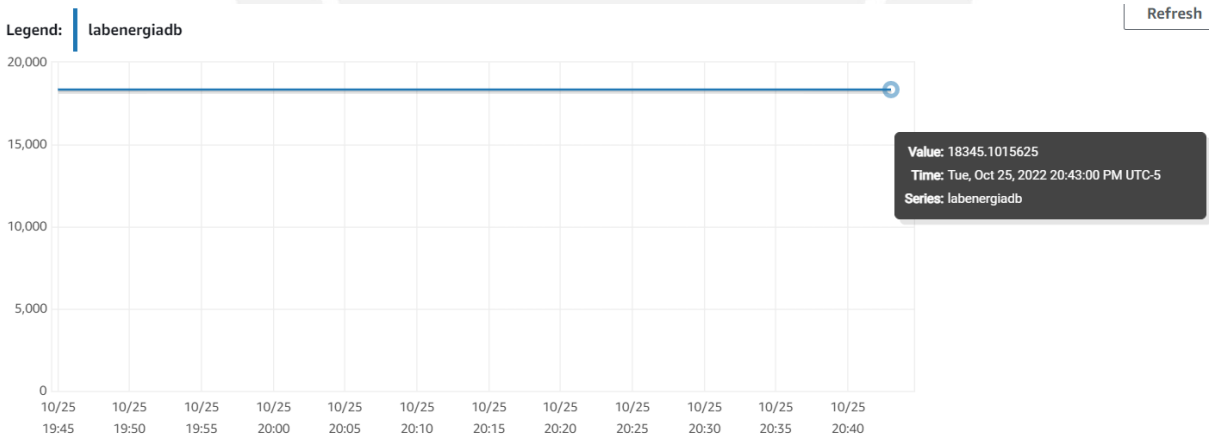


Figura 4.37 Espacio de almacenamiento libre después del laboratorio
Fuente: elaboración propia

Capítulo 5. Análisis de factibilidad técnica, económica, ambiental y social de la plataforma IoT desarrollada

5.1 Análisis de pertinencia técnica y tecnológicas

Desde un enfoque aplicativo y práctico, se puede notar que el diseño de la plataforma IoT desarrollada en la presente tesis es factible de ser implementada. Esto último quedó demostrado en la implementación de la misma y que la plataforma se encuentra actualmente en uso sin presentar problema alguno, como se describió en 4.8. Asimismo, la plataforma IoT desarrollada en la presente tesis significa el primer paso para que en un futuro la gran mayoría de los laboratorios de la universidad puedan realizarse de forma remota y se logre que la educación no presencial sea de calidad comparable a la presencial.

5.2 Análisis económico y financiero

Para analizar el costo de la plataforma implementada, se utilizó la documentación oficial de AWS, para saber qué aspectos de cada servicio utilizado tienen un costo, y la calculadora de

precios de AWS [58] para calcular el costo de cada servicio utilizado. El servicio OAuth 2.0 de Google Cloud Platform y Google Meet de Google Workspace no tienen costo. Los costos hallados son del tipo OPEX.

5.2.1 Costo de la instancia EC2

La instancia alquilada será del tipo "on demand", la cual tiene un costo por hora de uso. Las instancias EC2 tienen un costo de 0,0116 USD por hora. Así, la instancia cuesta aproximadamente 8,47 USD mensuales. Asimismo, el almacenamiento de la instancia tiene un costo. Se tiene el tipo de almacenamiento "SSD de propósito general", el cual tiene un costo de 0,10 USD por GB, lo que resulta en 1,60 USD por 16 GB de almacenamiento. Finalmente, el costo total mensual por el servicio EC2 de AWS resulta en aproximadamente 10,07 USD.

5.2.2 Costo del servicio IoT Core

El servicio de AWS IoT Core tiene dos principales costos: tiempo de conectividad de los dispositivos y cantidad de mensajes manejados. Según personal del Laboratorio de Energía, se tiene una cantidad promedio de 51 laboratorios que usan el banco de psicrometría, como puede observarse en la Tabla 5.1.

Tabla 5.1 Cantidad promedio de laboratorios por semestre
Fuente: personal del Laboratorio de Energía

Curso	Horarios	uso/horario	Sesiones
MEC245	10	3	30
MEC208	6	3	18
Extras	1	3	3
Total de sesiones			**51**

5.2.2.1 Costo por tiempo de conexión

Considerando que se tiene dos dispositivos a conectarse al IoT Core, el Servidor Web y el Raspberry Pi. Considerando 4 horas de duración del laboratorio, que son la cantidad de horas

que el Raspberry estará conectado, y que el Servidor Web estará conectado todo el día, se tiene 46860 minutos de conexión por mes. El costo de conexión por minuto es de 0,00000008 USD, lo cual resulta en un costo total de 0,0037 USD por mes.

Tabla 5.2 Costo promedio mensual por tiempo de conexión
Fuente: elaboración propia

Laboratorio	# sesiones	horas/laboratorio	
Psicrometría	51	4	
		204	Total horas Raspberry en 4 meses
		730	Total horas del servidor web por mes
		46860	Promedio de minutos por mes total
		0,0037	**Costo (USD)**

5.2.2.2 Costo de mensajería

La frecuencia de envío de *data* es de un mensaje por segundo por variable de entrada y se considera un fator de 10, obtenido de la experiencia (cada variable se utiliza aproximadamente 10 veces por laboratorio), para las variables de salida. Se obtiene 16162410 mensajes enviados al IoT Core, por tanto, se tiene la misma cantidad de mensajes enviados desde el IoT Core. Así, se tiene 32324820 mensajes por semestre, lo que resulta en 8081205 mensajes por mes. Finalmente, se obtiene un costo total de 8,08 USD por mes (el costo por millón de mensajes es de 1 USD).

Tabla 5.3 Cantidad promedio de mensajes enviados al bróker de IoT Core
Fuente: elaboración propia

Tipo de variable		#mensajes/sesion	Total mensajes/sesión	Total mensajes/semestre
# variables I	22	316800	316910	16162410
# variables O	11	110		

5.2.2.3 Costo total del uso de IoT Core

Sumando las cantidades calculadas en los dos puntos anteriores, el costo total del uso de IoT Core es de 8,08 USD.

5.2.3 Costo del servicio RDS

El costo del servicio RDS es similar al del EC2. La instancia alquilada será del tipo “on demand”, la cual tiene un costo por hora de uso. La instancia elegida tiene un costo 0,017 USD

por hora. Así, la instancia cuesta aproximadamente 12,41 USD mensuales. Asimismo, el almacenamiento de la instancia tiene un costo. Se tiene el tipo de almacenamiento "SSD de propósito general", el cual tiene un costo de 0,115 USD por GB, lo que resulta en 2,30 USD por 20 GB de almacenamiento. Finalmente, el costo total mensual por el servicio RDS de AWS resulta en aproximadamente 14,71 USD.

5.2.4 Costo del servicio Route 53

El costo por el nombre de dominio con el *Top-Level Domain* ".com" es de 12 USD anualmente y el costo de un *hosted zone* es de 0,5 USD mensualmente. Así, se tiene un costo mensual de 0.5 USD y otro costo de 12 USD cada 12 meses.

5.2.5 Flujo de caja anual

Con los costos calculados en los puntos anteriores se puede realizar el flujo de caja anual de la plataforma IoT implementada. La plataforma está implementada totalmente en la nube, por lo que no tiene costos de inversión CAPEX, sino todos los costos son del tipo OPEX.

Servicio	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
EC2	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07	-10,07
IoT Core	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08
tiempo de conexión	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
mensajería	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08	-8,08
RDS	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71	-14,71
Route 53	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-12,50
Hosted zone	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50	-0,50
Nombre de dominio	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-12,00
Flujo neto (USD)	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-33,36	-45,36

Figura 5.1 Flujo de caja anual
Fuente: elaboración propia

Asimismo, considerando una tasa de interés de 15% anual se obtiene un VAN (valor actual neto) de -381,95 USD. El proyecto es de ámbito social que beneficiará a la propia comunidad universitaria de la PUCP, por lo que la rentabilidad obtenida es del tipo social y todos los gastos serán cubiertos por el Laboratorio de Energía.

5.3 Análisis relacionado a la salud pública

La plataforma IoT implementada es una herramienta que puede ser utilizada en un contexto de pandemia y que ayudará a evitar el aumento del número de contagios que un contexto como ese significa, por lo que contribuye significativamente a proteger la salud pública en dicho contexto.

5.4 Análisis relacionado al bienestar y al orden social

La plataforma IoT implementada es una herramienta que puede ser utilizada, como se mencionó en 1.7.2, por estudiantes de la PUCP, sino que podrá ser utilizada por estudiantes que no cuenten con equipos de alto costo como el de banco de psicrometría. Así, se contribuye con la reducción de la brecha educativa vigente dentro del país, que, en la mayoría de casos, es por temas económicos.

5.5 Análisis ambiental y de sostenibilidad

Por el tema tratado, la tesis no justifica un análisis ambiental y de sostenibilidad.

CONCLUSIONES

1. Se logró diseñar e implementar una plataforma IoT que permita el monitoreo y control remoto desde la nube del banco de psicrometría del Laboratorio de Energía de la PUCP a un alumno a través de una interfaz web para la realización de laboratorios remotos.
2. La plataforma implementada permite la gestión de usuarios basado en los roles de alumno, jefe de práctica y administrador.
3. El acceso del rol alumno en la plataforma implementada se basa en fecha y horas, lo que permite lograr que un alumno no tenga acceso a la plataforma si no tiene un laboratorio asignado en la fecha y hora en la que intenta realizar su ingreso.
4. Se consiguió la visualización de la *data* generada por el equipo en tiempo real, mediante el uso del protocolo de comunicación MQTT y la tecnología WebSockets.
5. La plataforma permite la asignación de los interruptores que pueden ser controlados por un alumno sea en tiempo real mediante el uso de WebSockets.
6. La plataforma implementada permite la visualización a través de cámaras de aquellos medidores que no puedan generar *data* mediante la asignación de sus variables a sesiones de Google Meet.
7. Debido al éxito de las pruebas de rendimiento mostradas en el punto 4.5 y a las mejoras realizadas en el punto 4.6.2 a la plataforma implementada, esta ha sido utilizada a lo largo de un semestre académico por estudiantes sin presentar ningún inconveniente lo que demuestra que puede ser empleada para el desarrollo de laboratorios tanto en un contexto de presencialidad como en un contexto de no presencialidad.

BIBLIOGRAFÍA

[1] "PuntoEdu PUCP - Conoce los cursos PUCP disponibles para el semestre 2020-1." https://puntoedu.pucp.edu.pe/institucional/cursos-disponibles-2020-1/ (accessed Jul. 04, 2022).

[2] F. Palmieri, "Laboratorios virtuales en Ingeniería PUCP - PuntoEdu PUCP." https://puntoedu.pucp.edu.pe/vida-estudiantil/laboratorios-virtuales-en-ingenieria-pucp-se-desarrollan-con-exito/ (accessed Jul. 04, 2022).

[3] PUCP, "Conoce más del proceso de virtualización del Laboratorio de Energía." https://departamento.pucp.edu.pe/ingenieria/2021/09/02/conoce-mas-del-proceso-de-virtualizacion-del-laboratorio-de-energia/ (accessed Jul. 04, 2022).

[4] L. K. B. Félix, "DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PARA LA FASE DE CALENTAMIENTO Y ENFRIAMIENTO SENSIBLE DEL BANCO DE PSICROMETRÍA," PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ, 2010.

[5] "What Is the Internet of Things (IoT)?" https://www.oracle.com/internet-of-things/what-is-iot/ (accessed Jul. 07, 2022).

[6] M. Asemani, F. Abdollahei, and F. Jabbari, "Understanding IoT Platforms," *2019 5th Int. Conf. Web Res.*, pp. 172–177, 2019.

[7] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput. Commun.*, vol. 89–90, pp. 5–16, 2016, doi: 10.1016/j.comcom.2016.03.015.

[8] J. A. García Macavilca, "Estudio comparativo de plataformas IoT para el control y monitoreo remoto de un equipo mecánico para el envío y recepción datos," PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ, 2022.

[9] R. Pajuelo, "Covid-19: ministro de Salud anuncia el inicio de la cuarta ola en el Perú." https://andina.pe/agencia/noticia-covid19-ministro-salud-anuncia-inicio-de-cuarta-ola-el-peru-898964.aspx (accessed Jul. 07, 2022).

[10] K. M. Al-Aubidy, A. W. Al-Mutairi, and H. Z. Al-Kashashneh, "IoT Based Remote Laboratory for Solar Energy Experiments: Design and Implementation," in *18th IEEE International Multi-Conference on Systems, Signals and Devices, SSD 2021*, 2021, pp. 47–52. doi: 10.1109/SSD52085.2021.9429384.

[11] "Quiénes Somos - Red Peruana de Universidades." https://rpu.edu.pe/quienes-somos/#banner (accessed Jul. 07, 2022).

[12] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT,"

in *2018 IEEE International Conference on Future IoT Technologies, Future IoT 2018*, 2018, vol. 2018-Janua, pp. 1–8. doi: 10.1109/FIOT.2018.8325598.

[13] O. Toutsop, K. Kornegay, and E. Smith, "A Comparative Analyses of Current IoT Middleware Platforms," in *Proceedings - 2021 International Conference on Future Internet of Things and Cloud, FiCloud 2021*, 2021, pp. 413–420. doi: 10.1109/FiCloud49777.2021.00067.

[14] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," in *2016 Cloudification of the Internet of Things, CIoT 2016*, 2016, pp. 1–6. doi: 10.1109/CIOT.2016.7872918.

[15] A. Tamboli, *Build Your Own IoT Platform: Develop a Fully Flexible and Scalable Internet of Things Platform in 24 Hours*, 1st ed. Ed. Apress, 2019. doi: 10.1007/978-1-4842-8073-7.

[16] S. Khare and M. Totaro, "Big Data in IoT," *2019 10th Int. Conf. Comput. Commun. Netw. Technol. ICCCNT 2019*, pp. 6–12, 2019, doi: 10.1109/ICCCNT45670.2019.8944495.

[17] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, "Performance evaluation of CoAP and MQTT with security support for IoT environments," *Comput. Networks*, vol. 197, no. April, p. 108338, 2021, doi: 10.1016/j.comnet.2021.108338.

[18] I. E. T. F. IETF, "RFC 7252 - The Constrained Application Protocol - CoAP RFC 7252.pdf," *Request for Comments: 7252*. 2014. Accessed: Jun. 13, 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7252#section-2

[19] T. Jaffey, "MQTT and CoAP, IoT Protocols," *The Eclipse Foundation*, 2014. https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php (accessed Jun. 13, 2022).

[20] S. Hamdani and H. Sbeyti, "A comparative study of COAP and MQTT communication protocols," *7th Int. Symp. Digit. Forensics Secur. ISDFS 2019*, pp. 1–5, 2019, doi: 10.1109/ISDFS.2019.8757486.

[21] mqtt-v5.0, "MQTT Version 5.0," *Oasis-Open*, no. March, p. 137, Mar. 2019, Accessed: Jun. 13, 2022. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[22] MQTT, "MQTT: The Standard for IoT Messaging," *Mqtt.Org*, 2021. https://mqtt.org/ (accessed Jun. 13, 2022).

[23] P. Murley, Z. Ma, J. Mason, M. Bailey, and A. Kharraz, "Websocket adoption and the

landscape of the real-time web," in *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*, 2021, pp. 1192–1203. doi: 10.1145/3442381.3450063.

[24] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," *IEEE Internet Computing*, vol. 16, no. 4, IEEE, pp. 45–53, 2012. doi: 10.1109/MIC.2012.64.

[25] M. Tomasetti, "An Analysis of the Performance of Websockets in Various Programming Languages and Libraries," 2021. doi: 10.2139/ssrn.3778525.

[26] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in *2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016*, 2017, pp. 235–238. doi: 10.1109/ICACA.2016.7887957.

[27] S. Reetishwaree and V. Hurbungs, "Evaluating the performance of SQL and NoSQL databases in an IoT environment," in *2020 3rd International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering, ELECOM 2020 - Proceedings*, 2020, pp. 229–234. doi: 10.1109/ELECOM49001.2020.9297028.

[28] B. Anderson, "SQL vs. NoSQL Databases: What's the Difference? | IBM," *IBM Cloud*, 2021. https://www.ibm.com/cloud/blog/sql-vs-nosql (accessed Sep. 19, 2022).

[29] "Apache JMeter™," *Apache JMeter*, 2019. https://jmeter.apache.org/index.html (accessed Oct. 16, 2022).

[30] A. B. Bondi, "Characteristics of scalability and their impact on performance," *Proc. Second Int. Work. Softw. Perform. WOSP 2000*, pp. 195–203, 2000, doi: 10.1145/350391.350432.

[31] I. Ilunin, "How to Choose Your IoT Platform - Should You Go Open-Source?," *IoT All*, no. di, p. 1, 2017, Accessed: Jun. 20, 2022. [Online]. Available: https://medium.com/iotforall/how-to-choose-your-iot-platform-should-you-go-open-source-23148a0809f3

[32] IoT Analytics, "IoT Platform Companies Landscape 2021/2022," 2021. https://iot-analytics.com/iot-platform-companies-landscape/ (accessed Jun. 29, 2022).

[33] P. P. Ray, "A survey of IoT cloud platforms," *Futur. Comput. Informatics J.*, vol. 1, no. 1–2, pp. 35–46, 2016, doi: 10.1016/j.fcij.2017.02.001.

[34] B. Vogel, Y. Dong, B. Emruli, P. Davidsson, and R. Spalazzese, "What is an open IoT platform? Insights from a systematic mapping study," *Futur. Internet*, vol. 12, no. 4, pp. 1–19, 2020, doi: 10.3390/FI12040073.

[35] R. Goldman and R. P. Gabriel, "What Is Open Source?," in *Innovation Happens*

Elsewhere, 2005, pp. 29–73. doi: 10.1016/b978-155860889-4/50005-7.

[36] A. Tamboli, "Why Should You Build Your Own IoT Platform?," *tomorrow++*. https://medium.com/tomorrow-plus-plus/why-should-you-build-your-own-iot-platform-dff51578c0c (accessed Jul. 18, 2022).

[37] J. Bermúdez-Ortega, E. Besada-Portas, J. A. López-Orozco, and J. M. De La Cruz, "A new open-source and smart-device accessible remote control laboratory," in *Proceedings of 2017 4th Experiment at International Conference: Online Experimentation, exp.at 2017*, 2017, pp. 143–144. doi: 10.1109/EXPAT.2017.7984376.

[38] OpenJS Foundation, "Express - Node.js web application framework," *Expressjs.Com*, 2017. https://expressjs.com/ (accessed Sep. 16, 2022).

[39] G. C. Fernandez, E. S. Ruiz, M. C. Gil, and F. M. Perez, "From RGB led laboratory to servomotor control with websockets and IoT as educational tool," in *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2015, no. February, pp. 32–36.

[40] Socket.IO, "Introduction | Socket.IO," 2022. https://socket.io/docs/v4/ (accessed Sep. 16, 2022).

[41] P. Jitthammapirom, P. Chayratsami, and W. Somha, "Development of Remote Laboratory for Feedback Control System Class," in *6th International STEM Education Conference, iSTEM-Ed 2021*, 2021, pp. 10–13. doi: 10.1109/iSTEM-Ed52129.2021.9625131.

[42] J. A. Sanchez-Viloria, L. F. Zapata-Rivera, C. Aranzazu-Suescun, A. E. Molina-Pena, and M. M. Larrondo-Petrie, "Online Laboratory Communication Using MQTT IoT Standard," in *Proceedings of 2021 World Engineering Education Forum/Global Engineering Deans Council, WEEF/GEDC 2021*, 2021, no. November, pp. 550–555. doi: 10.1109/WEEF/GEDC53299.2021.9657292.

[43] iot.eclipse.org, "Eclipse Mosquitto," *Eclipse Mosquitto*, 2018. https://mosquitto.org/ (accessed Sep. 16, 2022).

[44] A. O. Kurniawan, A. R. Hakiki, K. N. Banjarnahor, M. A. Hady, A. Santoso, and A. Fatoni, "Internet Based Remote Laboratory Architecture for 3-Phase Induction Motor Control System Experiment," in *Proceedings - 2021 International Seminar on Intelligent Technology and Its Application: Intelligent Systems for the New Normal Era, ISITIA 2021*, 2021, pp. 381–385. doi: 10.1109/ISITIA52817.2021.9502222.

[45] Amazon Web Services, "AWS IoT Core Documentation," 2019. https://docs.aws.amazon.com/iot/ (accessed Oct. 24, 2022).

[46] G. Ongo and G. P. Kusuma, “Hybrid Database System of MySQL and MongoDB in Web Application Development,” in *Proceedings of 2018 International Conference on Information Management and Technology, ICIMTech 2018*, 2018, no. September, pp. 256–260. doi: 10.1109/ICIMTech.2018.8528120.

[47] Amazon Web Services Inc, “What Is Amazon Relational Database Service (Amazon RDS)? - Amazon Relational Database Service,” 2017. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html (accessed Oct. 24, 2022).

[48] A. W. S. AWS, “What Is Amazon EC2? - Amazon Elastic Compute Cloud,” *awsdocumentation*, 2014. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (accessed Oct. 24, 2022).

[49] AWS, “AWS IoT Core policies - AWS IoT Core,” *AWS*, 2021. https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html (accessed Oct. 24, 2022).

[50] “Passport.js.” http://www.passportjs.org/ (accessed Oct. 23, 2022).

[51] “Getting Started | Fast-CSV.” https://c2fo.github.io/fast-csv/docs/introduction/getting-started (accessed Oct. 19, 2022).

[52] “PM2, Node.js Production Process Manager with a built-in Load Balancer.” https://github.com/Unitech/pm2 (accessed Oct. 15, 2022).

[53] Amazon Web Services Inc, “What Is Amazon Route 53? - Amazon Route 53,” 2017. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/Welcome.html (accessed Oct. 22, 2022).

[54] A. W. S. Inc, “Working with public hosted zones - Amazon Route 53.” https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/AboutHZWorkingWith.html (accessed Oct. 22, 2022).

[55] NGINX Inc., “Welcome to NGINX Wiki’s documentation!,” *NGINX Wiki*, 2015. https://www.nginx.com/resources/wiki/ (accessed Oct. 23, 2022).

[56] Let’s Encrypt, “Getting Started - Let’s Encrypt.” https://letsencrypt.org/getting-started/ (accessed Oct. 23, 2022).

[57] Electronic Frontier Foundation, “About Certbot.” https://certbot.eff.org/pages/about (accessed Oct. 23, 2022).

[58] AWS, “AWS Pricing Calculator,” *AWS*, 2019. https://calculator.aws/#/ (accessed Oct. 30, 2022).