UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

“DISEÑO Y DESARROLLO DE UN PROTOTIPO DE MONITOREO DE OXÍGENO UTILIZANDO TECNOLOGÍA IOT E INTELIGENCIA ARTIFICIAL”

Trabajo de titulación previo a la obtención del

Título de Ingeniero Electrónico

AUTOR: OLIVER STEEVEN VANEGAS CÁCERES.

TUTOR: ING. DIEGO FREIRE QUIROGA.

Guayaquil-Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Oliver Steeven Vanegas Cáceres con documento de identificación N° 0927682989 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 22 de octubre de 2024

Atentamente,

Oliver Steeven Vanegas Cáceres

0927682989

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Oliver Steeven Vanegas Cáceres con documento de identificación No. 0927682989, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto técnico: “Diseño y desarrollo de un prototipo de monitoreo de oxígeno utilizando tecnología IoT e inteligencia artificial”, el cual ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 22 de octubre de 2024

Atentamente,

Oliver Steeven Vanegas Cáceres
0927682989

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Diego Roberto Freire Quiroga con documento de identificación N° 0917208084, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DISEÑO Y DESARROLLO DE UN PROTOTIPO DE MONITOREO DE OXÍGENO UTILIZANDO TECNOLOGÍA IOT E INTELIGENCIA ARTIFICIAL, realizado por Oliver Steeven Vanegas Cáceres con documento de identificación N° 0927682989, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 22 de octubre de 2024

Atentamente,

Ing. Diego Roberto Freire Quiroga Msc.
0917208084

DEDICATORIA

Este proyecto de titulación va dedicado en primer lugar a Dios, por darme sabiduría y poder culminar esta carrera, sobre todo dedico de manera muy especial este proyecto a mis padres, Antonio Vanegas Pacheco y Cecilia Cáceres Toledo quienes sacrificándose de muchas maneras me motivaron siempre y que todo ese esfuerzo y sacrificio que realizaron en mi ahora los llene de orgullo. También dedico este éxito a mi esposa Jamileth e hijo Kael Sebastián, por estar siempre conmigo e impulsarme a seguir adelante. Por último, quiero dedicar este proyecto a mis hermanos Martín, Jonathan e Israel quienes con sus ejemplos me impulsaron a seguir adelante y a saber que en esta vida “todo pasa”.

Oliver Steeven Vanegas Cáceres

AGRADECIMIENTO

Quiero agradecer en primer lugar a Dios por darme la sabiduría necesaria para lograr este logro académico, a cada miembro de mi familia en especial a mis queridos Padres que fueron mi apoyo de manera moral y económica durante todo este largo camino, agradezco también a mi esposa e hijo por ser mi inspiración en todo momento.

A mi tutor Msc. Diego Freire quien estuvo presente en todo momento impartiendo su conocimiento y experiencia para lograr este proyecto de titulación.

Oliver Steeven Vanegas Cáceres.

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO TÉCNICO	TEMA DE PROYECTO TÉCNICO
2024	• OLIVER STEEVEN VANEGAS CÁCERES	Ing. Diego Freire Quiroga, Msc.	Diseño y desarrollo de un prototipo de monitoreo de oxígeno utilizando tecnología IoT e inteligencia artificial

El proyecto propone crear un prototipo de monitoreo de niveles, para visualizar la cantidad de oxígeno medicinal que mantiene un tanque que se encuentra conectado a un paciente con insuficiencia respiratoria, permitiendo así la visualización del manómetro de manera remota, logrando tener monitoreado tanto el nivel oxígeno que mantiene el tanque, como la cantidad de oxígeno que se está suministrando al paciente. La lectura de los manómetros se realizará con la inteligencia artificial mediante una cámara Raspberry la cual estará realizando el reconocimiento de los distintos niveles que maneje el manómetro, la cámara que estará visualizando el manómetro podrá reconocer en que ángulo se encuentra la perrilla insufladora y así poder detectar a qué nivel se encuentra el tanque de oxígeno y a la misma vez saber qué cantidad de oxígeno se está suministrando al paciente.

Toda la programación de la inteligencia artificial se realizará en el módulo Raspberry Pi4 mediante el sistema operativo Raspbian. Este dispositivo mediante su editor de código OpenCV y su lenguaje de programación de Python, será el encargado de almacenar toda la información necesaria incluyendo la información captada por la cámara que será enviada al sistema de NODE RED la cual se encargará de enviar los datos a la nube y así poder ser visualizada en el aplicativo web por la persona encargada en ese momento del cuidado del paciente.

Además, el prototipo incluye un sistema de alarma en el aplicativo web, que alertará al personal médico cuando los niveles de oxígeno están por debajo de un umbral crítico, permitiendo así el reemplazo a tiempo y oportuno del tanque de oxígeno. Este sistema de alarma sirve de gran ayuda al personal médico, en muchas ocasiones existen casos de emergencias con otros pacientes, lo que dificulta avanzar a donde se encuentra el tanque y verificar si aún mantiene oxígeno disponible, por ese motivo se realizó este sistema de alarma, que emite una señal de alerta cuando la perilla insufladora llega al inicio del umbral más bajo.

PALABRAS CLAVES: MANÓMETROS, PERILLA INSUFLADORA, NUMPY, VISUAL STUDIO CODE, VAADIN, AZURE, OPEN CV, RASPBERRY PI 4, NUMPY, CÁMARA WEB, OXÍGENO.

ABSTRACT

YEAR	STUDENTS	DIRECTOR OF TECHNICAL PROJECT	TECHNICAL PROJECT THEME
2024	• OLIVER STEEVEN VANEGAS CÁCERES	Ing. Diego Freire Quiroga, Msc.	Diseño y desarrollo de un prototipo de monitoreo de oxígeno utilizando tecnología IoT e inteligencia artificial

The project proposes to create a level monitoring prototype, to visualize the amount of medicinal oxygen stored by a tank that is connected to a patient with respiratory failure, allowing the visualization of the pressure gauge remotely, achieving monitoring of both the oxygen level stored by the tank, and the amount of oxygen being delivered to the patient. The reading of the pressure gauges will be carried out with artificial intelligence using a Raspberry camera which will be recognizing the different levels that the pressure gauge handles, the camera that will be viewing the pressure gauge will be able to recognize at what angle the insufflation knob is located in order to detect at what level the oxygen tank is and at the same time know what amount of oxygen is being supplied to the patient.

All artificial intelligence programming will be carried out on the Raspberry Pi4 module using the Raspbian operating system. This device, through its OpenCV code editor and its Python programming language, will be in charge of storing all the necessary information, including the data captured by the camera that will be sent to the NODE RED system, which will be in charge of sending the data to the cloud and thus be able to be viewed in the web application by the person in charge of the patient's care at that moment.

In addition, the prototype includes an alarm system in the web application, which will alert medical personnel when oxygen levels are below a critical threshold, allowing timely replacement of the oxygen tank. This alarm system is a great help to medical personnel, on many occasions there are cases of emergencies with other patients, which makes it difficult to go to where the tank is located and verify if it still has oxygen available, for this reason this alarm system was created, which emits an alert signal when the insufflation knob reaches the beginning of the lowest threshold.

KEY WORDS: MANOMETER, NUMPY, VISUAL STUDIO CODE, VAADIN, AZURE, OPEN CV, RASPBERRY PI 4, NUMPY, WEB CAMARA, OXÍGEN.

ÍNDICE GENERAL

ÍNDICE DE FIGURA

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la mayoría de los hospitales no existe un sistema de control inteligente para conocer qué cantidad de oxígeno manejan los tanques, y también existe una demanda alta de pacientes con respecto al personal médico, por ese motivo, se dificulta tener un control de los pacientes que presentan insuficiencia respiratoria y los cuales están conectados a un tanque de oxígeno.

El proceso anteriormente descrito implica que el personal de enfermería este constantemente realizando un control de la lectura del manómetro cuando el médico de cabecera lo solicite, por lo cual, si la lectura estaba en parámetros críticos recién se realizaban las medidas necesarias corriendo el riesgo de haberse quedado sin oxígeno y no darse cuenta en el momento preciso.

Este método no es el más seguro debido a que si el nivel baja puede ocasionar daños irreparables a los pacientes, incluso llegando hasta la muerte. El presente proyecto de titulación busca ofrecer una solución a todas las situaciones anteriormente mencionadas, se va a implementar un algoritmo con inteligencia artificial para monitorear y controlar el nivel de oxígeno en el tanque, a su vez, se realizará un aplicativo móvil para poder visualizar los niveles de oxígeno de los diferentes tanques y recibir alertas cuando los niveles estén bajos.

Una característica fundamental del prototipo es su capacidad para emitir alertas automáticas en el aplicativo web, las cuales notificarán al personal médico cuando los niveles de oxígeno estén por debajo de un umbral crítico. Esta funcionalidad no solo mejora la eficiencia en la gestión de los recursos, sino que también garantiza una respuesta rápida y eficaz ante situaciones de emergencia, asegurando así la continuidad del suministro de oxígeno sin interrupciones que puedan afectar la atención al paciente.

En síntesis, este proyecto representa un avance significativo en la tecnología aplicada a la salud, proporcionando herramientas innovadoras para mejorar la calidad de vida de pacientes con insuficiencia respiratoria y facilitar el trabajo del personal médico en la gestión de recursos críticos como el oxígeno medicinal.

1 EL PROBLEMA

1.1 Antecedentes

Este prototipo de monitoreo de oxígeno pretende reducir la sobre carga de trabajo al personal médico, que muchas veces tiene a su recaudo pacientes con insuficiencia respiratoria, al momento de verificar de manera presencial en qué nivel de oxígeno se encuentra el tanque y si está suministrando la cantidad de oxígeno requerido, se tiende a descuidar a los demás pacientes o viceversa; al momento de medicar a otros pacientes se suele descuidar al enfermo que está conectado a un tanque de oxígeno, y esto es un gran problema porque puede llegar a sufrir de hipoxia que es la falta de oxígeno en la sangre o sufrir de hipoxemia que es la falta de este gas al momento de formar tejidos, otro inconveniente es al realizar el llamado para que se realice el cambio de tanque de oxígeno, existe desconocimiento si la persona encargada de realizar el cambio, estará ocupada realizando el reemplazo de otro tanque en otra área o estará en otro trabajo que le dificulte realizar el cambio de tanque en ese preciso momento, por esa razón es necesario realizar el llamado a la persona encargada de dicho trabajo con unos minutos de anticipación, para evitar cualquier tipo de inconveniente.
Desde muchos años atrás y hasta la actualidad el único método de control y monitoreo de un tanque de oxígeno medicinal era la visualización de manera presencial del personal de enfermería, eso implica que cada personal que se encuentre de turno y tenga un paciente o persona que sufra de insuficiencia respiratoria y esté conectado a un tanque de oxígeno tendrá que avanzar hasta donde se encuentra el tanque para verificar a qué nivel se encuentra y verificar si es el momento indicado para cambiar de tanque por otro que se encuentre con un nivel de oxígeno más alto.

1.2 Importancia

En estos tiempos el uso de la inteligencia artificial en el área de programación es una herramienta importante porque permite diagnosticar de forma más rápida y precisa la información por la cual es entrenada.

Al implementar esta inteligencia artificial para la lectura de niveles de oxígeno ayudará tanto al paciente como al personal médico, porque habrá un mejor cuidado en el oxígeno otorgado al paciente, evitando que él mismo se quede sin suministro de oxígeno.

La implementación de este nuevo sistema de monitoreo tendrá los siguientes beneficios:

- Monitoreo constante de los niveles de oxígenos.
- Evitará el error humano en las lecturas del manómetro que están en el tanque de oxígeno.
- Se evitará que el tanque de oxígeno llegue a su nivel más bajo emitiendo una alerta de bajo nivel de oxígeno.

1.3 Alcance

El alcance del presente proyecto de titulación es:

- **Diseño del Prototipo de Monitoreo del Tanque de Oxígeno:** Este incluiría la selección de los componentes necesarios, como sensores de flujo y manómetros, así como la configuración física del hardware para permitir la lectura precisa de los niveles de oxígeno en el tanque.

- **Desarrollo del Algoritmo de Visión Artificial:** Este objetivo implica la creación de un algoritmo que pueda interpretar las lecturas de los manómetros y el flujo del oxígeno del tanque mediante análisis de imágenes capturadas por una cámara, lo que permitirá monitorear el suministro de oxígeno al paciente en tiempo real.

- **Desarrollo del Aplicativo móvil para Monitoreo:** Aquí se llevará a cabo el diseño y la implementación de una aplicación móvil que recibirá datos del prototipo de monitoreo y los presentará de manera clara y accesible para los usuarios. Esto puede incluir visualizaciones gráficas de los niveles de oxígeno en el tanque y la cantidad suministrada al paciente, así como alertas en caso de niveles críticos.

1.4 Delimitación

1.4.1 Espacial

Esta propuesta de tesis será un prototipo de monitoreo de oxígeno y será aplicado en un tanque de oxígeno dentro de las instalaciones de Universidad Politécnica Salesiana como se muestra en la **Figura 1.**

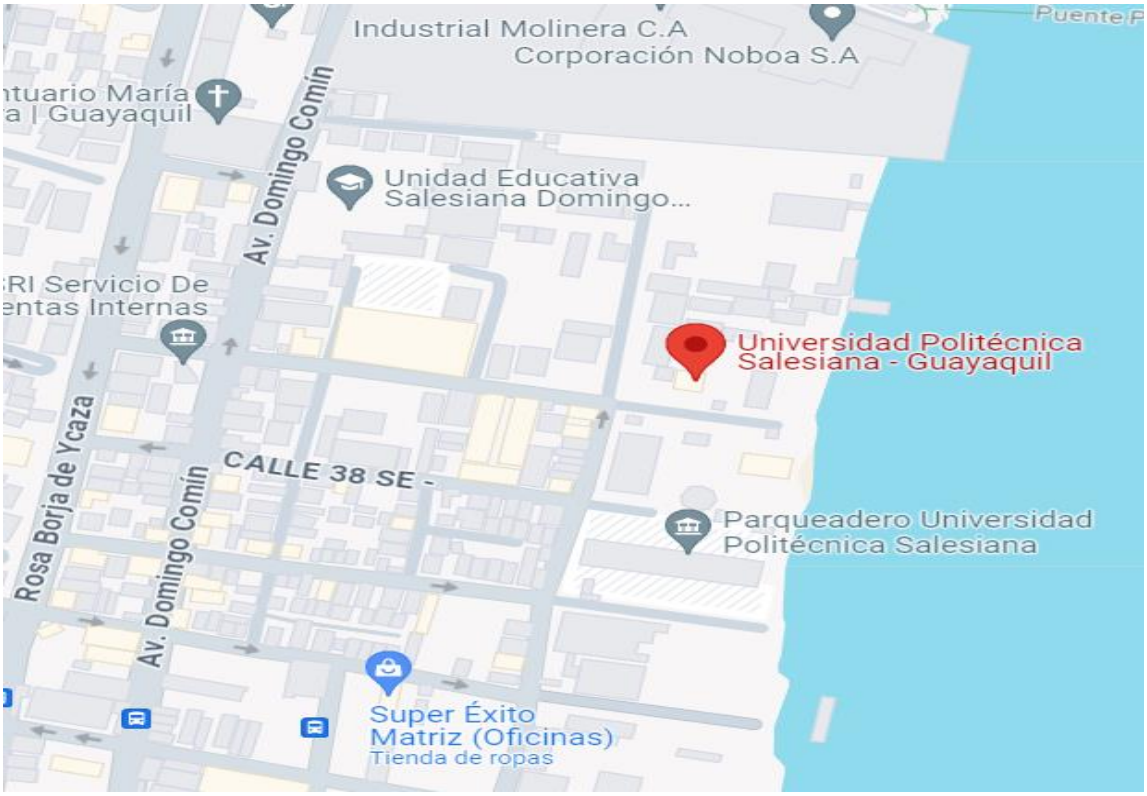


Figura 1. Ubicación de Universidad Politécnica Salesiana (Google maps, 2008).

1.4.2 Temporal

El presente proyecto de titulación se efectuó en el periodo de junio del 2024 a agosto del 2024.

1.4.3 Académica

El presente tema de titulación consistió en crear una plataforma de control con inteligencia artificial de bajo costo usando Raspberry Pi 4, una cámara web compatible con la Raspberry a utilizar, motivo por el cual es necesario contar conocimientos sólidos en la parte de programación, redes inalámbricas y electrónica.

Adicional se debe contar con conocimientos en el desarrollo de aplicaciones móviles para poder realizar la implementación de un aplicativo para la visualización de los niveles de oxígenos, los programas más usados para realizar aplicaciones móviles son Android Studio, flutter, mit app inventor, etc.

1.5 Explicación del problema

El control preciso en los tanques de oxígeno es crucial, porque al no tener la cantidad de oxígeno adecuado que requiere el paciente, esto ocasionará aumento en el ritmo cardiaco, hipoxia cerebral y así ocasionar rápidamente la muerte o daño cerebral grave. La instauración aguda de la hipoxia durante más de 4 minutos conduce a la parada cardiorrespiratoria y la afectación irreversible de múltiples órganos vitales. Las células más afectadas son las neuronas, miocárdicas, túbulos renales y las hepáticas (PASCUAL, s.f.).

El método de monitoreo actual con el que cuenta el hospital es la verificación presencial, el cual consta que el personal médico encargado vaya a realizar una lectura de manera presencial al manómetro cuando el médico lo solicite, esto conlleva a no tener un control constante de los niveles de oxígenos.

Por descuido de atender una emergencia en muchas ocasiones se tiende a descuidar los tanques de oxígenos que están conectados a los pacientes que mantienen insuficiencia respiratoria, esto causaría un gran problema para el paciente el cuál correría el riesgo de sufrir un paro cardio-respiratorio por falta de oxigenación en el cuerpo.

Si se desea contratar más personal hospitalario para cada paciente, se necesitaría más presupuesto económico cada año, el cual muchas veces los hospitales públicos no cuentan con este presupuesto requerido.

2 OBJETIVOS

2.1 Objetivo General

Diseñar, desarrollar e implementar un prototipo que sirva para monitorear el flujómetro y el manómetro de un tanque cuando está suministrando oxígeno a una persona integrando la inteligencia artificial y el sistema IoT para el envío de datos.

2.2 Objetivo específico

- Diseñar el prototipo para el monitoreo del flujómetro y manómetro de un tanque de oxígeno presente.

- Desarrollar un algoritmo de visión artificial para monitorear el oxígeno suministrado al paciente mediante la lectura del manómetro y flujómetro del tanque.

- Desarrollar un aplicativo web para el sistema de monitoreo para la visualización de oxígeno en el tanque y cantidad de oxígeno suministrada al paciente.

- Realizar pruebas al sistema en condiciones reales, para validar su precisión y seguridad.

3 FUNDAMENTOS TEÓRICOS

3.1 Tipos de gases medicinales

Los gases medicinales en la oxigenoterapia representan un punto muy importante al momento de ingresar un paciente para ser atendido en una sala de un hospital, ya que sus características son empleadas para el consumo humano y aplicaciones medicinales, con el objetivo de prevenir, prescribir, tratar, aliviar y/o curar enfermedades que la persona padezca (Chavez, 2022).

3.1.1 Oxígeno O_2.

El oxígeno es el gas medicinal más ampliamente utilizado y responde a las exigencias establecidas para cualquier medicamento en relación a la eficacia, seguridad y calidad (Dominguez Gil, 2005).

Es por esta razón que se han creado algunas formas de disminuir y atender estos casos de cuadros respiratorios que las personas puedan llegar a tener; entre las principales esta la oxigenoterapia. La cual se encarga de brindar oxígeno, con propósitos curativos para elevar la cantidad de oxígeno inspirada “fio2” en un enfermo. El gas medicinal es brindado al paciente por medio de cánulas nasales o mascarillas, y para lograr brindar este gas medicinal hay

estructuras de almacenamiento de oxígeno, como pueden ser dispositivos de suministro de oxígeno o sistemas de administración de oxígeno.

Como se muestra en la **Figura 2.** los concentradores de oxígeno tienen como fin contrarrestar o tratar fallas respiratorias que consiste en problemas en el intercambio gaseoso de la sangre, pero para poder llegar a las instancias actuales con la tecnología que soporta la terapia de oxígeno hubo una pequeña evolución (David, 2017).

Figura 2. Oxígeno (Alonso, 2021).

3.1.2 Óxido Nitroso N_2O

Como se muestra la **Figura. 3** el N2O, con no menos del 21% de O2, es utilizado para la inducción y mantenimiento de la anestesia general, en combinación con otros anestésicos, relajantes musculares, analgésicos, etc (Dominguez Gil, 2005).

Figura 3. Óxido nitroso N_2O (INFRA DE HONDURAS, 2020)***.***

Hace muchos años atrás el óxido nitroso fue revelado por Joseph Priestley quien era experto en ciencias y además fue el que descubrió el oxígeno. El experto en química Humphrey Davy en el año de 1798 encuentra las propiedades analgésicas del óxido nitroso, este descubrimiento lo realizó cuando respiró óxido nitroso para lograr hallar o experimentar sus resultados, luego de sentir dolencias en un diente que se encontraba con erupciones parciales; luego de algunos años, el mismo experto en química Davy difundió una publicación en donde se llegó a concluir que el óxido nitroso podría llegar a tener ventajas al momento de realizar alguna cirugía (yarzabal et al., 2018).

3.1.3 Aire medicinal

El aire es la combinación de gases que no se pueden ver, ni oler y tampoco. El aire se encuentra estructurado de manera primordial por una combinación de 78% de nitrógeno y 21% oxígeno como se muestra en la **Figura. 4.** Obteniendo residuos de Helio, Kriptón, Argón, Hidrógeno, así como vapor de agua. En esencia se usan en los hospitales como fluido motriz o elemento respiratorio (Abásolo, 2005).

Figura 4. Gas medicinal (Universo, 2021)***.***

3.2 Tarjeta de memoria SD

Una tarjeta de memoria SD es un dispositivo que sirve para almacenar información de manera digital, una ventaja de la tarjeta de memoria es que son portátiles y se pueden trasladar a cualquier lado, en el mercado existen tres formatos de tarjeta de memoria en donde su única diferencia son sus medidas. La estructura más grande es el SD estándar, luego continua el formato miniSD y por último el microSD. Las tarjetas SD tienen 9 pines, uno sirve para enviar comandos, tres son de alimentación, cuatros sirven para datos y uno es de reloj.

Como se muestra en la **Figura 5.** la tarjeta de memoria está estructurada de una manera que pueda ser compatible con los discos duros. La tarjetas están compuestas por 8 bits, los cuales se reúnen en bloques, que son grupos de 512 bytes, La unidad básica es el byte. El bloque es la unidad básica de memoria que se puede escribir o leer de una tarjeta (Pérez et al ., 2023).

Figura 5. Tarjeta de memoria microSD (Pérez et al ., 2023)

3.3 Cámara Web

Una cámara web o webcam como se muestra en la **Figura 6.** es una pequeña cámara digital conectada a una computadora, la cual puede capturar imágenes y transmitirlas a través de internet, ya sea a una página web u otras computadoras de forma privada.

Figura 6. Cámaras Web (Moreno García, 2014).

Para que se puedan transmitir las imágenes las cámaras web necesitan de un computador. Aunque existen otros tipos de cámara que solo necesitan conectarse a la red informática, ya sea por cable de red o de manera inalámbrica. Para diferenciarlas de la webcam o cámaras de web se las denomina net cam o cámaras de red (Chávez et al ., 2009).

3.4 Regulador de presión

Como se muestra en la **Figura 7.** el regulador de presión mide la presión a la que se encuentra el oxígeno dentro del cilindro, mediante una aguja sobre una escala graduada (Montalvo López, 2010).

Figura 7. Regulador de presión (David, 2017).

3.5 Regulador de flujo

También conocido como manorreductor, regula la presión a la que le sale O_2 del cilindro y permitir saber la cantidad de oxígeno que se está administrando por minuto (litros/minuto), esto se indica sobre una escala graduada como se muestra en la **Figura 8.** (Montalvo López, 2010)**.**

Figura 8. Regulador de flujo (David, 2017).

3.6 Raspberry pi

La Raspberry Pi ha sido construida específicamente como una computadora y altamente flexible a una fracción del costo de una PC tradicional para que cualquier persona pueda utilizarla para resolver problemas de manera creativa. Como se muestra en la **Figura 9.** su gran cantidad de recursos supera fácilmente sus limitaciones y convierte a Raspberry Pi en una excelente herramienta de investigación que puede usarse para casi cualquier cosa (Jolles, 2021).

Figura 9. Raspberry Pi (Jolles, 2021).

Desde su primer lanzamiento en 2012, se han lanzado varias generaciones de computadoras Raspberry Pi, que se pueden clasificar en tres modelos distintos: Raspberry Pi A, B y Zero (un cuarto modelo, el Compute Module, se utiliza principalmente en aplicaciones industriales) (Jolles, 2021).

3.7 Lenguaje de programación que utiliza Raspberry

3.7.1 Python

Python es un lenguaje de programación que sirve como una alternativa primordial para el desarrollo de programa libre porque cumple con los requisitos planteados.

Como se muestra en la **Figura 10.** Python tiene una manera muy fácil de realizar programaciones orientada a objetos, cuenta con facilidades para la programación orientada a objetos, necesaria y práctica, por lo que se estima un lenguaje multiparadigmas.

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello World" <<
endl;
return 0;
}
```

Figura 10. Programa Hola Mundo en C++ (Challenger et al ., 2014).

El lenguaje de programación de Python fue inventado al finalizar los años 80 por un programador holandés Guido Van Rossum, en el momento que estaba trabajando en Amoeba, que es un sistema operativo. Primariamente se concibe para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC. Python 2.0 fue creado en el mes de octubre del año 2000, el soporte completo a Unicode y la recopilación de basura eran las principales características que contenía Python 2.0. Guido Van Rossum realizó el mayor avance, logrando ser verdaderamente implementado por las personas. En el año 2008 aparece Python 3.0 que es una versión mejor e incompatible con las versiones pasadas en diferentes facetas, aparece el 3 de diciembre en el mercado luego de un tiempo largo de estudio. Algunas de las cualidades introducidas en la versión 3 han logrado ser compatibles en la versión 2.6 para realizar de forma más fácil la transición entre estas. El premio del software libre fue entregado al programador Guido van Rossum en el 2001, por sus labor en

la creación y desarrollo del lenguaje Python. En el 2005 fue contratado por Google, donde trabaja en la actualidad, aunque sigue liderando los esfuerzos en el desarrollo del Python (Challenger et al ., 2014).

3.8 Raspbian

Raspberry utiliza el sistema operativo de Raspbian el cual cuenta con más de 35000 paquetes, optimizados para lograr un mejor funcionamiento en Raspberry Pi, En junio de 2012 fue terminado Raspbian, pero aún se encuentra en activo desarrollo con una insistencia en lo que se refiere a mejorar la estabilidad y la productividad de la mayor cantidad de paquetes como sea posible. Es necesario decir que para lograr la comunicación serial entre el PIC y Raspberry Pi® se ha configurado el puerto serial de esta computadora embebida (Quintana et al ., 2015).

3.9 Servicios en la nube

Como se muestra en la **Figura 11.** Existen tres grandes grupos de servicios en la nube:

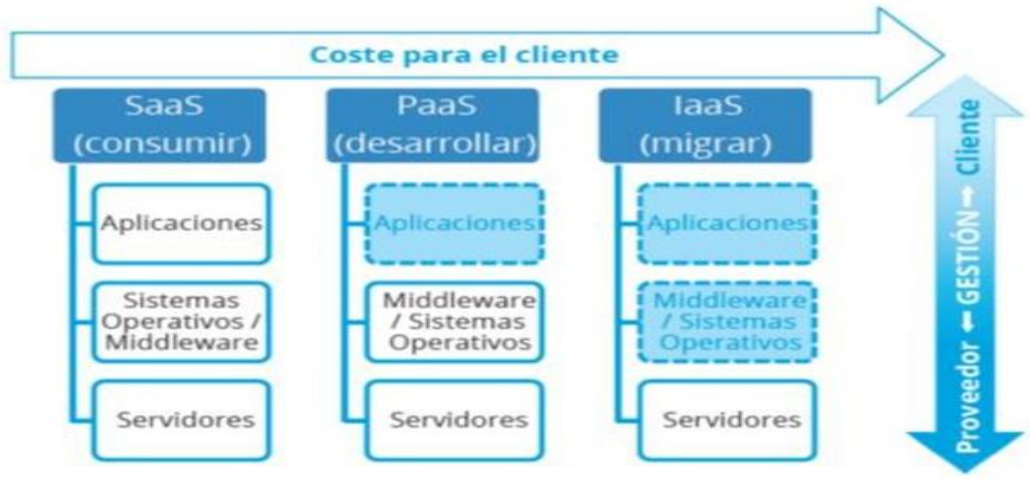


Figura 11. SaaS vs PaaS VS IaaS (Rala, 2021).

IaaS (Infrastructure as a Service): Es el diseño más complicado, pero también es el modelo que más dominio le entrega al cliente. En esta infraestructura, la herramienta más utilizada por el cliente es conseguido por medio del proveedor externo del servicio. Este servicio es regularmente utilizado por directores de sistemas, pues brindan comodidades, como poder elegir la cantidad de almacenamiento y escoger el sistema operativo.

PaaS (*Platform as a Service*): En este modelo, a diferencia del *IaaS*, el cliente no tiene el control sobre la infraestructura de almacenamiento o de las redes, sino que es el propio proveedor el que proporciona todo lo necesario que llevar a cabo el desarrollo de las aplicaciones desde su desarrollo hasta su puesta en producción. Este esquema es muy utilizado por los ingenieros de software ya que su única función es enfocarse en la implementación del código. Ejemplos de este tipo de servicio son *Heroku* o *CloudFountry*.

SaaS (*Software as a Service*): En este servicio la persona o cliente solo tendrá acceso al programa creado por una compañía que se encuentra alojado en la nube. La responsabilidad de la empresa es brindar al cliente el mantenimiento, desarrollo, soporte y operaciones del software. En este servicio, al cliente debe importarle únicamente como se usa el servicio y no estar pendiente del proceso de mantenimiento o por cuestiones técnicas como la infraestructura del servicio. *Microsoft Office 365* o las aplicaciones web de *Google* son ejemplos de este tipo de servicio.

3.9.1 Microsoft Azure

Microsoft Azure es la infraestructura de computación en nube pública de Microsoft. Brindando una galería de servicios en la nube, incluidos los de computación que son analítica, almacenamiento y redes como se muestra en la **Figura 12**. Las personas pueden escoger entre estos servicios para programar y escalar nuevas aplicaciones, o trabajar en aplicaciones que ya existen, en la nube pública. Azure fue creado por Microsoft en octubre de 2008.

Figura 12. Computación en la nube (Rala, 2021).

En abril del 2014 Windows Azure como se llamaba de manera original paso a tener el nombre de Microsoft Azure. Amazon Web Services y Google Cloud Platform son las competencias con respecto a plataforma de nube publica con las Azure compite hoy en día. En todo el mundo Microsoft tiene centros de datos de Azure, para asegurar una mejor disponibilidad. A comienzos del año 2016, los servicios de Azure están accesibles en 22 regiones del mundo, incluyendo en Brasil, Europa, Asia, Estados Unidos y Australia. Azure utiliza un modelo de precios de pago 'como va', que se carga basado en el uso, esto ocurre de igual manera con otros proveedores de nube pública. Sin embargo, una sola aplicación puede utilizar múltiples servicios de Azure, por lo que los usuarios deben revisar y administrar el uso para minimizar los costos (Baquero, 2020).

Azure tiene otros servicios como permitir desplegar servicios de infraestructuras como redes, almacenamiento, máquinas virtuales y plataformas como CMS para desarrollo de Web, backend para aplicaciones móviles, bases de datos de alta disponibilidad SQL compatibles con todo tipo de tecnología, como por ejemplo bases de datos Oracle, Node Js, Máquinas Linux, MySQL, etc. El punto más fuerte es la virtualización, pero si además se utiliza Active Directory y Windows 10 la empresa puede reservar tiempo de gestión, costos y capacidad gracias a la capacidad de gestionar nuestra organización sin recurrir a un servidor de dominio tradicional on premise. Al utilizar una identidad única podemos integrar nuestro directorio activo con Enterprise Mobility + Security y Microsoft Office 365, donde los usuarios pueden iniciar todas sus aplicaciones en la nube por medio de un panel de acceso completamente personalizado basado en web o por medio de una app móvil y tener la misma experiencia de uso, sin importar ni el equipo ni el sistema operativo (Baquero, 2020).

3.10 Node Red

Node-RED es un recurso de programación visual el cual brinda la facilidad al usuario de programar sin escribir código, ya que expone visualmente las relaciones y las funciones. Está basado en navegador y permite insertar, eliminar y conectar entre sí nodos con el fin de que se comuniquen entre ellos. Se efectuó a inicios del 2013, a partir de un proyecto del Grupo de Servicios de Tecnologías Emergentes entre Nick O'Leary y Dave Conway-Jones. Todo empezó con la prueba de un concepto para lograr ver y manipular topics de MQTT que poco a poco fue obteniendo mejores resultados hasta lograr convertirse en un proyecto que se extendió para varias direcciones. En septiembre de 2013 se convirtió oficialmente en una

herramienta de código abierto. En la actualidad es una de las herramientas que más se usan en el ámbito de IoT por las siguientes razones:

- Esta mantenida por el International Business Machines Corporation y tiene código abierto.
- Se puede utilizar con el navegador y esto facilita el trabajo con cualquier sistema operativo.
- Es muy intuitivo y logra crear prototipos rápidamente sin tener que profundizar en técnicas de programación.
- Se puede utilizar de manera simultánea en varios procesos por lo que evita las tareas repetitivas.
- Es un sistema libre y gratuito.

Node-RED se basa en el concepto Flow-Bassed Programming como se puede visualizar en la ***Figura 13.*** La codificación basada en flujos es una forma de programación a partir de una red de nodos. Se tratan de una especie de cajas negras en las que se ocultan código de Node.js y tienen un propósito diferenciado. Por ejemplo, un nodo captura los datos y los envía a otro nodo para que seleccione algunos de ellos mediante un filtro (Miguel, 2021).

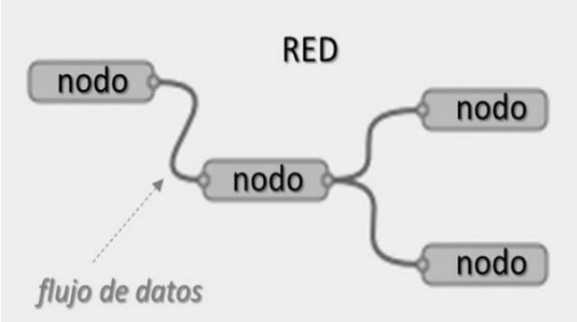


Figura 13. Red y flujo de datos entre nodos (Miguel, 2021).

3.10.1 Tipos de Nodos

Node red maneja diferentes tipos de nodos, entre los más importante están:

- **Iniciadores:** Como su nombre lo indica son aquellos nodos que inician un flujo de datos, el único puerto que poseen es el de salida y pueden ser de dos tipos:

- Activos: Como lo indica su nombre se activan de una manera periódica bajo la supervisión de un evento determinado. Por ejemplo, que se envíe la hora actual cada 2 segundos (Miguel, 2021)

- Pasivos: Es el caso de MQTT. Este nodo está a la espera de que se actualice un topic en el broker, y cuando lo realiza recibe este dato actualizado (Miguel, 2021).

- **Intermedios:** Estos nodos tienen dos puertos uno de entrada y otro de salida. Estos nodos que reciben un flujo de datos, realizan las funciones pertinentes dependiendo del nodo que se trate y generan una salida con el resultado de esa operación. En este tipo de nodos, cabe hacer referencia al “fuction node”, el cual nos permite incorporar directamente código propio de JavaScript para implementar nuestra propia función (Miguel, 2021).

- **Terminales:** Este tipo de nodos tienen una sola salida y es utilizada para realizar eventos y ejecutar otros flujos de aplicación como por ejemplo la entrega de datos al bróker, estos nodos se sitúan al final de las ramas del flujo de datos. (Miguel, 2021).

Por defecto, en Node-RED tienen incorporados los siguientes nodos como se muestra en la **Figura 14.**

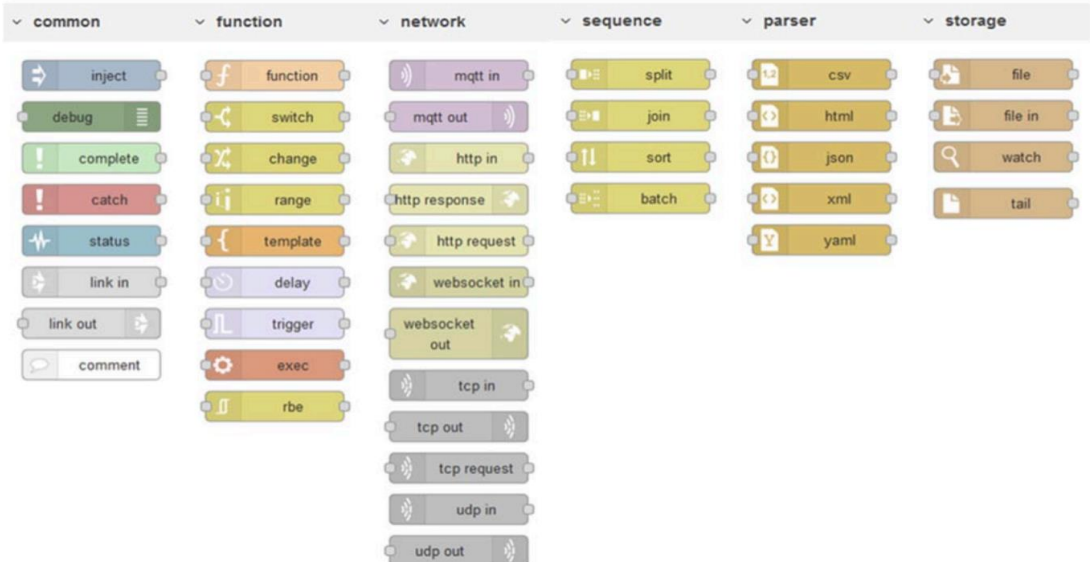


Figura 14. Nodos por defectos en Node-Red (Miguel, 2021).

3.11 Visual Studio Code

El VSC cuenta actualmente con la función de detectar el lenguaje de programación de cada archivo, a través de una extensión. Por ejemplo, los archivos con extensión.blade.php son asociados a código HTML con directivas del motor blade de Laravel. Los archivos con extensión .php son asociados con código PHP. Una vez detectado el lenguaje en el que está escrito cada archivo, VSC marca de manera gráfica con color rojo, los errores de sintaxis para que el programador los identifique y los pueda corregir rápidamente. Asimismo, VSC es capaz de identificar cuando se abre una función, o ciclo, o bucle de programación y marca en rojo cuando no se detecta el cierre correspondiente para su corrección. Además de esta funcionalidad propia del VSC, se pueden agregar complementos o extensiones para apoyar la tarea de programación de componentes y sistemas, tales como los siguientes: Run on Save. Revisa estilos de codificación y eficiencia de las líneas codificadas en el momento de guardar el archivo del módulo, de tal manera que elimina cosas innecesarias, como líneas en blanco excesivas y propone cierto grado de refactorización en el código, cambiando las líneas de código, o eliminando variables innecesarias, por ejemplo; PHP Intelephense. Realiza autocompletado de código, mostrando en una pequeña ventana emergente diferentes opciones de selección, ante una instrucción PHP. Lo anterior apoya al programador para tener a la mano la estructura y sintaxis de funciones básicas de PHP (Romero, 2023).

3.12 Framework Vaadin

Vaadin es un Framework que sirve para crear innovadoras aplicaciones web, con una característica atractiva e intuitiva y de magníficas prestaciones. Además, este Framework ayudará al desarrollo de la aplicación de una manera más eficiente mediante sus propias hojas de estilo denominada Syntactically Awesome Stylesheets, las mismas que son una extensión de CSS ya que admite dos modelos de programación tanto para el cliente como el servidor, ayudando a reducir el tiempo de desarrollo y el número de errores (Guachamin et al ., 2017).

3.12.1 Características de Framework Vaadin

Como se muestra en la **Figura 15.** Existen algunas características de Framework Vaadin.

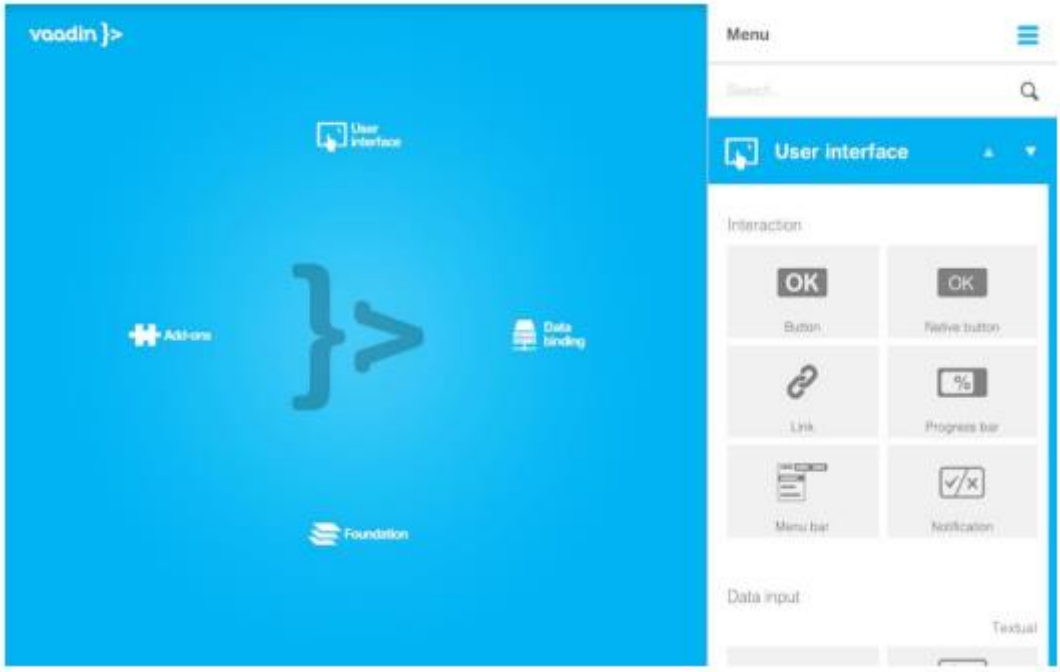


Figura 15. Marco de trabajo de Vaadin (Guachamin et al ., 2017).

- Marco Integral del Componente soporta el patrón MVC incluso tiene consigo una colección de componentes como widgets personalizados y atractivos (Guachamin et al ., 2017).

- Ajusta el aspecto que tiene temas y estilos personalizar sin modificar el código java (Guachamin et al ., 2017)

- Construcción de Aplicaciones Web Seguras que dirige la situación de la interfaz de usuario del servidor, aparte tiene el código de aplicación, lógica empresarial y validaciones en el servidor, la cual tiene una autenticación segura de parámetros y requerimientos (Guachamin et al ., 2017)

- Realiza trabajos con implementos como plugin Eclipse IDEA, IntelliJ IDEA, aparte de realizar integración con NetBeans IDE entre otras (Guachamin et al ., 2017)

- Se adapta a los importantes navegadores tanto en audio como en video y no es necesario tener compatibilidad Web (Guachamin et al ., 2017)

- Utiliza el lenguaje de programación java para el desarrollo web e incluso es orientado a objeto, esto lo vuelve seguro y fácil. (Guachamin et al ., 2017).

- Arquitectura de componentes extensible como widgets personalizados que se pueden construir con: GWT, JavaScript, WebComponents (Guachamin et al ., 2017).

4 MARCO METODOLÓGICO

4.1 Diseño del prototipo

Como se muestra en la **Figura 16.** para la ejecución del proyecto del sistema de monitoreo de un manómetro se tomó la decisión de usar un sistema embebido Raspberry pi 4 como controlador del sistema, con un sistema operativo llamado raspbian. Este proyecto cuenta con diferentes etapas, en primera instancia se encuentra la etapa de reconocimiento de imagen, donde se obtiene la imagen de los niveles de oxígeno presentado en el manómetro, luego continuamos con la etapa de adquisición de valores y mediante la IA se realiza el procesamiento de datos para posteriormente ser enviada a una base de datos mediante protocolo de comunicación “Nodo de red”.

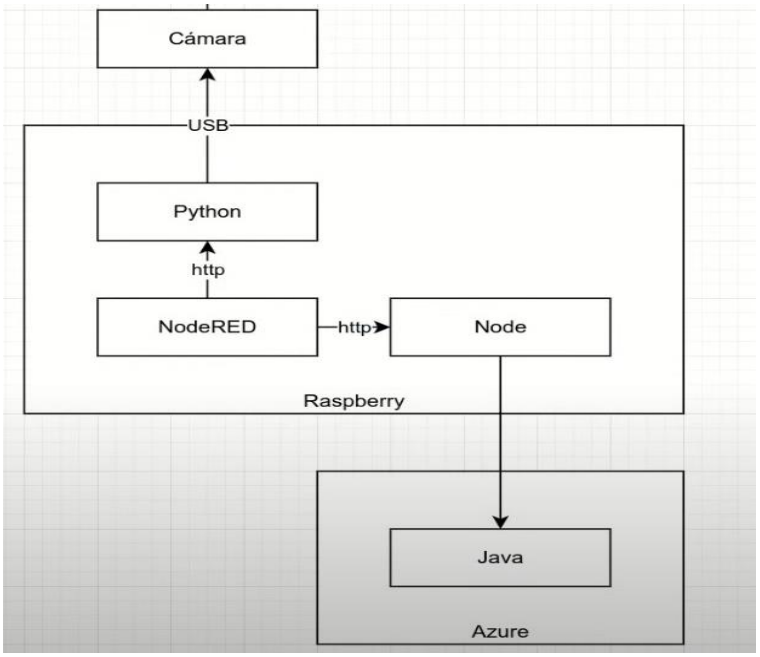


Figura 16. Diseño del prototipo de monitoreo.

4.2 Etapa de adquisición de datos

4.2.1 Raspberry y Cámara web

En la **Figura 17.** se detalla la conexión entre la Raspberry que se está utilizando y la cámara web, como se puede visualizar en la conexión se realiza mediante cable USB.

Figura 17. Cámara web con Raspberry Pi (GEOVANNY, 2019).

4.2.2 Inicialización y Adquisición de datos

Para poder utilizar este tipo de cámara en nuestra Raspberry pi se realizará la importación de la librería CV2, como se puede observar en la **Figura 18.** se inicializa con una variable llamada camera que va a captar la información de nuestra cámara conectada vía USB, si por algún motivo la cámara no pudo inicializar con éxito, retornará un mensaje indicando que no se pudo abrir la cámara.

```
USB_CAMERA = '/dev/video2'
camera = cv2.VideoCapture(USB_CAMERA)
if not camera.isOpened():
    return print('camera not opened')
```

Figura 18. Código de la importación de la librería CV2.

Una vez se haya realizado correctamente la inicialización de la adquisición de datos para la cámara, como se muestra en la **Figura 19.** se crea un lazo **while** infinito para estar siempre

leyendo la información obtenida por la cámara. Para poder obtener los valores de la cámara se coloca el comando **camera.read().**

```
while True:
    try:
        last_measurements = []
        for index in range(0, 10):
            retry, frame = camera.read()
            if not retry:
                print('error getting frame from camera')
                camera.release()
                return

            print('computing degrees')
            if len(sys.argv) == 1:
                show_images = False
            else:
                show_images = sys.argv[1] == 'true'
            image, measurements = recognizement.get_degrees_multiple(index, frame, show_images=show_images, last_right=last_right, last_left=last_left)
```

Figura 19. Código que permite leer la información visualizada por la cámara.

4.3 Etapa de procesamiento de datos

Para poder realizar un correcto reconocimiento de valores del manómetro se debe realizar algunos cambios en la imagen obtenida por la cámara, entre los cambios se encuentra cambiar a escala de grises la imagen capturada como se muestra en la **Figura 20.** este cambio se realiza mediante el comando **cvtColor ()** que se encuentra en la librería CV2 como se muestra en la **Figura 21.**

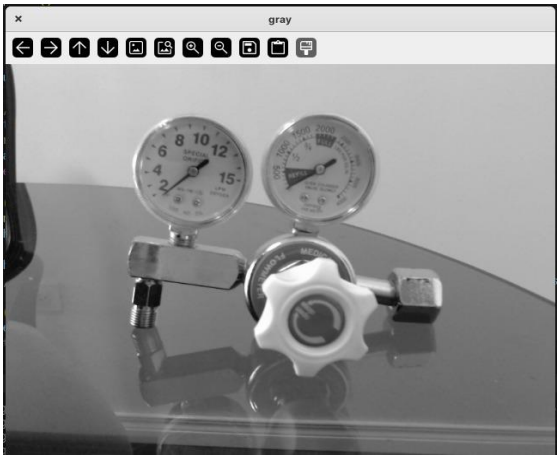


Figura 20. Foto del manómetro en escala de grises.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
if show_images:
    show_allow_to_close('gray', gray)
```

Figura 21. Comando para cambiar a escalas de grises la imagen.

Luego, se procede a recortar la imagen para trabajar con la parte circular del manómetro. Como se muestra en la **Figura 22.** sobre la imagen recortada se logra un reconocimiento circular que serían los bordes de nuestros manómetros representados con color verde. Para lograr este reconocimiento de bordes se debe trabajar con el comando **HoughCircles()** como se muestra en la **Figura 23.**

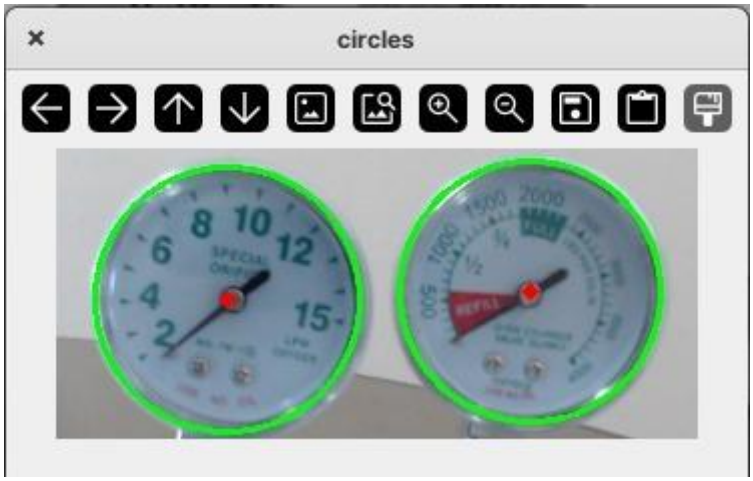


Figura 22. Reconocimiento de bordes.

```
_, width = blur.shape[:2]
max_radius = int(0.9 * (width / 2) / 2)
min_radius = int(0.5 * (width / 2) / 2)

circles = cv2.HoughCircles(image=blur,
                           method=cv2.HOUGH_GRADIENT,
                           dp=1.2,
                           minDist=2 * min_radius,
                           param1=50,
                           param2=50,
                           minRadius=min_radius,
                           maxRadius=max_radius)
```

Figura 23. Código para reconocimiento de bordes.

Para realizar el reconocimiento de valores se debe identificar el centro del manómetro y proceder a eliminar bordes y valores numéricos como se muestra en la **Figura 24.**

Figura 24. Reconocimiento del centro del manómetro.

Se debe definir una función que ayude con la eliminación de los bordes y de la parte numérica. Esta función se llama crop_circle y se inicia con “def” palabra clave que usa Python para definir una función como se muestra en el código que se encuentra la **Figura 25.** Esta función está diseñada para recortar una porción circular de una imagen, utilizando las coordenadas y el radio especificados en el parámetro circle, las coordenadas son x, y, r. La opción show_images puede ser útil para depurar o visualizar los resultados del recorte.

```python
def crop_circle(image, circle, show_images = False):
    (x, y, r) = circle
    extra_space = -int(0.5 * r)
    top_left_x = max(0, x - r - extra_space)
    top_left_y = max(0, y - r - extra_space)
    bottom_right_x = min(image.shape[1], x + r + extra_space)
    bottom_right_y = min(image.shape[0], y + r + extra_space)

    image = image[top_left_y:bottom_right_y, top_left_x:bottom_right_x]
    cropped = image.copy()
    if show_images:
        show_allow_to_close('croped', cropped)
    return cropped
```

Figura 25. Código para la eliminación de los bordes y parte numérica del manómetro.

A continuación, se procede con la eliminación de los colores verdes y rojos de la **Figura 24.** ya que solo es necesario trabajar con la flecha de la perilla insufladora y su inclinación, para poder realizar la eliminación de los colores se crea dos funciones que ayudaran a poder realizar dichas acciones, como se muestra en el código de la **Figura 26.** Los comandos a utilizar para remover los colores son remove_green () para el color verde y remove_red () respectivamente.

```
def remove_green(image, show_images = False):
    green_channel = image[:, :, 1]
    threshold = 90
    mask = green_channel > threshold
    image[mask] = [255, 255, 255]
    if show_images:
        show_allow_to_close('no green', image)

def remove_red(image, show_images = False):
    red_channel = image[:, :, 2]
    threshold = 60
    mask = red_channel > threshold
    image[mask] = [255, 255, 255]
    if show_images:
        show_allow_to_close('no red', image)
```

Figura 26. Código para la eliminación del color ver y rojo del manómetro.

El resultado de deshacerse de los colores verde y rojo se muestra en la ***Figura 27.*** Y solo se logra visualizar la perilla insufladora del manómetro.

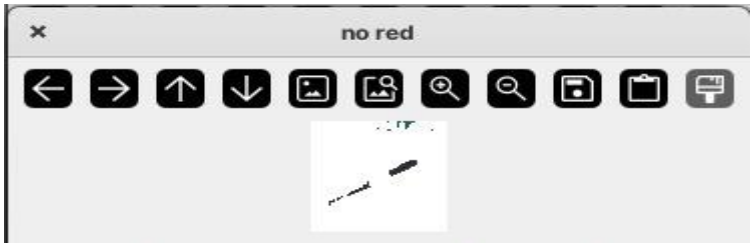


Figura 27. Perilla insufladora sin color verde y rojo.

Luego se detecta los bordes de la flecha y se realiza un agrandamiento de los bordes para tener mejor visualización como se muestra en la **Figura 28.**

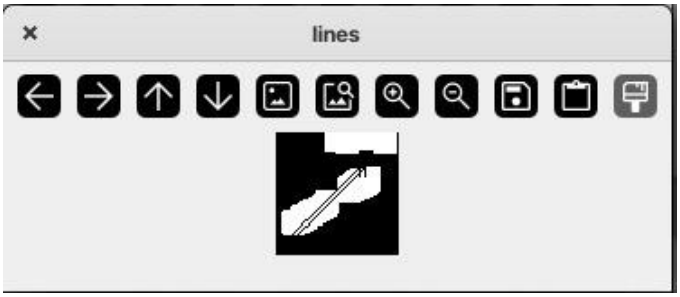


Figura 28. Detección de los bordes.

Una vez detectado los bordes se procede a trazar líneas, para lograr descifrar de mejor manera la que se parezca a la línea de la perilla insufladora. Como se muestra en la **Figura 29.** la función **def get_line()** sirve para procesar líneas y coordenadas de manera significativa, como por ejemplo iterar sobre líneas y a su vez seleccionar una línea especifica y resaltarla, en este caso la que se parezca a la perilla insufladora.

```
def get_line(image, lines, h, xmax1s, ymax1s, xmax2s, ymax2s, show_images):
    a = len(h)
    h.sort(reverse=True)
    m = 0
    k = 0

    xmax1 = 0
    xmax2 = 0
    ymax1 = 0
    ymax2 = 0
    xs1 = 0
    xs2 = 0
    ys1 = 0
    ys2 = 0
```

Figura 29. Procesamiento de líneas y coordenadas.

Una vez concluida la obtención de la línea de la perilla, se procede a realizar un cálculo matemático para obtener el ángulo de inclinación y a su vez parametrizar los valores reales del manómetro.

Luego se realiza un código en Python usando el método matemático del teorema de Pitágoras para poder calcular el ángulo de inclinación de la perilla como se muestra en la **Figura 30.** El propósito de este código es calcular el ángulo que una línea definida por el punto (xminutes, yminutes) forma con el eje vertical (y) en el centro de una imagen. Este cálculo es relevante en aplicaciones como la detección de la dirección de objetos en imágenes, análisis de movimiento, o cualquier otra aplicación que requiera conocer la orientación de una línea respecto al centro de una imagen.

```
            if (m == h[0]):
                if (hypo2 == h[f]):
                    if ((math.sqrt((xmax2 - x2)**2 + (ymax2 - y2)**2)) > 20):
                        if ((math.sqrt((xmax1 - x1)**2 + (ymax1 - y1)**2)) > 20):
                            xs1 = x1
                            xs2 = x2
                            ys1 = y1
                            ys2 = y2
                            cv2.line(image, (xs1, ys1), (xs2, ys2), (0, 255, 0), 3)
                            k = 1
                            break
        if (k == 1):
            break

    height, width = image.shape[:2]
    xcenter = width / 2
    ycenter = height / 2

    hassona1 = abs(xcenter - xs1)
    hassona2 = abs(xcenter - xs2)
    abdo1 = abs(xcenter - xmax1)
    abdo2 = abs(xcenter - xmax2)
    if (hassona1 > hassona2):
        xhours = xs1
        yhours = ys1
    else:
        xhours = xs2
        yhours = ys2

    if (abdo1 > abdo2):
        xminutes = xmax1
        yminutes = ymax1
    else:
        xminutes = xmax2
        yminutes = ymax2

    #calculating theta for hours hand
    l1 = math.sqrt(((xcenter - xhours)**2) + ((ycenter - yhours)**2))
    l2 = ycenter
    l3 = math.sqrt(((xcenter - xhours)**2) + ((0 - yhours)**2))
    cos_theta_hours = (((l1)**2) + ((l2)**2) - ((l3)**2)) / (2 * (l1) * (l2))

    if show_images:
        show_allow_to_close('line', image)

    return cos_theta_hours, xcenter, xhours, xminutes, ycenter, yminutes
```

Figura 30. Obtención del ángulo de la perilla insufladora.

4.4 Etapa de envió de datos

4.4.1 Implementación en Python del endpoint para la solicitud de datos

Se empieza con la importación de la librería flask como se muestra en la **Figura 31.** que sirve para el desarrollo de aplicaciones web en Python. Es ligero y flexible, adecuado para proyectos pequeños y medianos, y también se puede escalar para aplicaciones más grandes con las extensiones adecuadas.

```
 8   import cv2
 9   import numpy
10   import flask
11
12   app = flask.Flask(__name__)
13
```

Figura 31. Importación de la librería flask.

Luego se procede a realizar un endpoint para ser usado con node-red cuando sea necesario. Como se muestra en la **Figura 32.** este endpoint lo que realiza es un llamado a la función

get_measurements() la cuál manejará solicitudes POST a la ruta /api/measurements. Para obtener los datos el endpoint intenta capturar imágenes de una cámara para realizar alguna operación, en este caso mediciones. Si no puede acceder a la cámara o capturar imágenes, la función registra el error y responde con un estado de error HTTP 500.

```python
@app.post('/api/measurements')
def get_measurements():
    global last_right
    global last_left

    try:
        app.logger.info('defining input')
        camera = cv2.VideoCapture(0)

        app.logger.info('opening camera')
        if not camera.isOpened():
            app.logger.error('camera not opened')
            return flask.jsonify({}), 500

        last_measurements = []
        for index in range(0, 10):
            retry, frame = camera.read()
            if not retry:
                app.logger.error('error getting frame from camera')
                camera.release()
                return flask.jsonify({}), 500
```

Figura 32. Definición del endpoint.

Dentro de la función **get_measurements()** existe una función principal que se llama **get_degrees_multiple()**, esta función forma parte de la clase **recognizement()** y lo que realiza es la obtención de datos en crudo de nuestros dos manómetros. Como se muestra en la **Figura 33.** el código está registrando el inicio de un proceso de cálculo de grados y luego llama a una función para realizar dicho cálculo, almacenando los resultados en la variable measurements. Este proceso es parte del sistema de reconocimiento de imágenes o video.

```python
app.logger.info('computing degrees')
_, measurements = recognizement.get_degrees_multiple(index, frame, show_images=False, last_right=last_right, last_left=last_left)
```

Figura 33. Código para la obtención de datos de los dos manómetros.

Luego se realiza la separación de datos de acuerdo al tipo de información que recibe, como se muestra en la **Figura 34.** el proceso de este código es identificar y almacenar la última medición de cada posición ´derecha o izquierda´ según corresponda. Cada medición que se va registrando queda almacenada en un historial o base de datos.

```
    app.logger.info('separing measures')
    for measurement in measurements:
        if measurement['position'] == 'left':
            last_left = measurement
        elif measurement['position'] == 'right':
            last_right = measurement

    last_measurements += measurements

app.logger.info('computing average of measurements')
letf_average = recognizement.get_averaged(last_measurements, 'left')
right_average = recognizement.get_averaged(last_measurements, 'right')

app.logger.info('getting measurements')
left = [measurement for measurement in last_measurements if measurement['position'] == 'left' and measurement['value'] == letf_average]
right = [measurement for measurement in last_measurements if measurement['position'] == 'right' and measurement['value'] == right_average]

app.logger.info('grouping final measures')
to_loop = []
if len(left) > 0:
    to_loop.append(left[0])
if len(right) > 0:
    to_loop.append(right[0])
```

Figura 34. Código para la separación de mediciones de acuerdo a su posición.

Una vez obtenido los datos de manera separada, se convierte la lista to_loop a una cadena JSON utilizando **json.dumps()** Como se muestra en la **Figura 34.** esto se realiza mediante la librería numpy para transformar nuestros datos en crudo en formato JSON.

```
            app.logger.info(a)

        return json.dumps(to_loop, cls=recognizement.NumpyEncoder)
    except Exception:
        tb = traceback.format_exc()
        app.logger.error(tb)
```

Figura 35. Devolución de resultados en formato json.

4.4.2 Implementación de protocolo Nodo de Red

Como se muestra en la **Figura 36.** Se crea un algoritmo de 5 bloques dentro del software Node-RED:

1. **Every second:** Este nodo está configurado para inyectar automáticamente un mensaje en el flujo cada segundo. Se usó para la actualización de datos en tiempo real de nuestro monitoreo, el cuál necesita ejecutarse de manera continua y periódica.

2. **Check manometer:** desde este bloque se manda a ejecutar el reconocimiento, el cual está destinado a solicitar los datos procesados a la Raspberry con respecto a la función del monitoreo del manómetro

3. **Debug 2:** este bloque permite la visualización de los datos recibidos mediante el bloque “Check manometer”.

4. **http request:** realiza el cambio de formato de los datos recibidos y los envía al servidor.

5. **Debug 1:** este bloque permite la visualización de datos que son enviados al servidor.

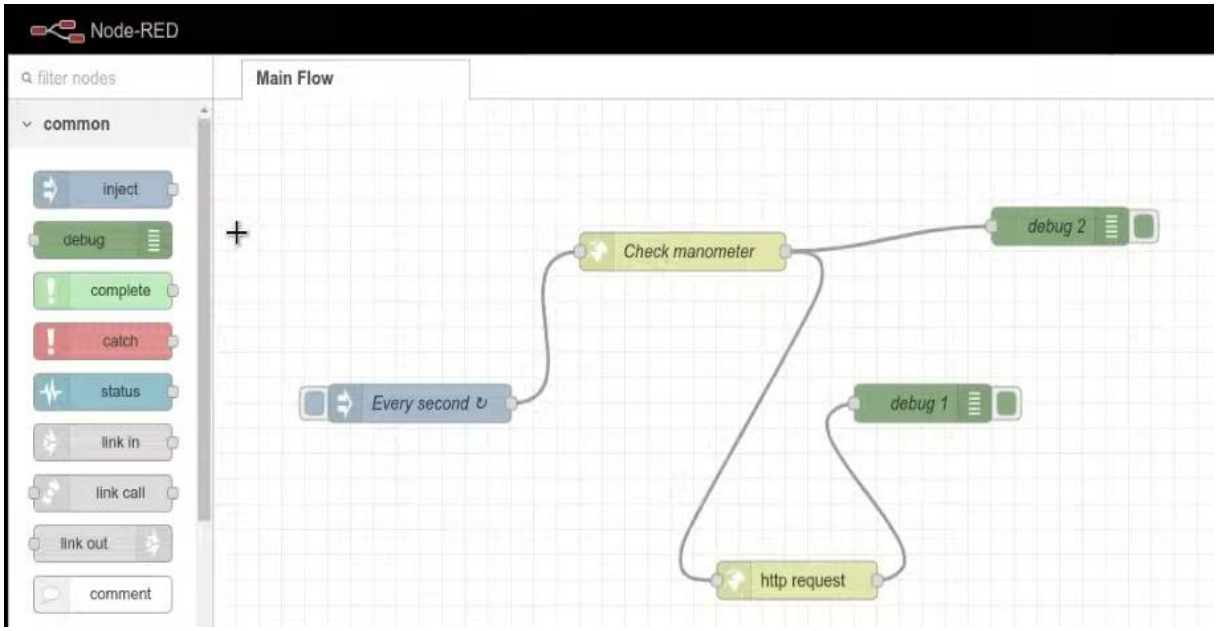


Figura 36. Algoritmo de 5 bloques.

Como se muestra en la **Figura 37.** Dentro del bloque Check manometer se coloca el tipo de método http que se realizará, en este caso es un método POST, este método se lo va a utilizar para la aplicación web porque sirve para enviar datos al servidor. ya que retornará información de los datos procesados de la cámara; En la URL se coloca el endpoint previamente definido en el código realizado en Python para que pueda comunicarse con la Raspberry.

Figura 37. Configuración del bloque Check manometer.

El tipo de dato una vez realizada la solicitud del bloque Check manometer está en formato JSON, como se puede apreciar en la **Figura 38.**


```
"[{"position": "left", "value":
2057.2660914052112, "origin":
[613, 217], "destination": [618,
143]}, {"position": "right",
"value": 2.9780340881507175,
"origin": [220, 280],
"destination": [160, 312]}]"
```


Figura 38. Información de los manómetros en formato json.

Este tipo de dato es procesado por Node red y enviado a otro servicio en este caso se creó un Node para que los datos se puedan enviar a la máquina virtual y posteriormente al servidor web. Como se muestra en la **Figura 39.** este es el correcto formato que debe tener para que el servidor pueda entender los datos que le llegan.

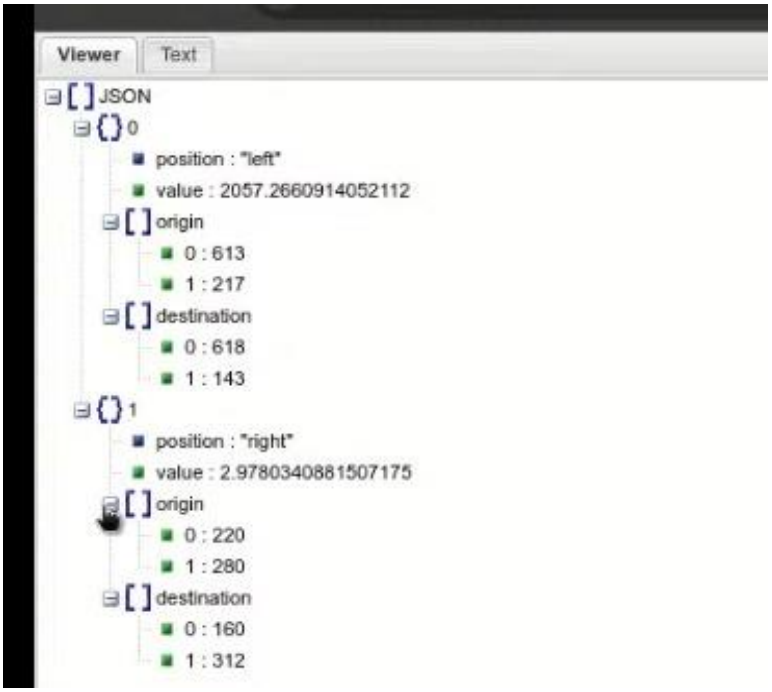


Figura 39. Información que será enviada al servidor web.

4.4.3 Implementación de comunicación entre Node-Red y Máquina Virtual Azure

En primer lugar, se debe importar las diferentes librerías que se van a usar, en este caso la más importante es Jmix que sirve para proporcionar un conjunto de herramientas y marcos para construir aplicaciones empresariales complejas de manera más eficiente, está librería en esta ocasión ayudará a realizar el cambio de formato, luego como se muestra en la **Figura 40.** Se importa el módulo Http que es utilizado para el servidor web y para manejar las solicitudes de comunicación Http.

```
import './env'
import Jmix from './libraries/jmix'
import Http, {IncomingMessage, ServerResponse} from 'http'
```

Figura 40. Importación de librerías para la comunicación entre Node-Red y la máquina virtual.

Las variables declaradas son utilizadas para pasar parámetros a la instancia de Jmix. Las variables que se muestran en la **Figura 41.** permiten que la configuración de la aplicación se separe del código fuente, lo que es útil para poder manejar la configuración en los entornos de desarrollo, pruebas, producción. En pocas palabras este código ayudará a interactuar con

una API o sistema externo. Al utilizar estas variables, el código se vuelve más flexible y adaptable a nuestro entorno sin necesidad de cambiar el código fuente.

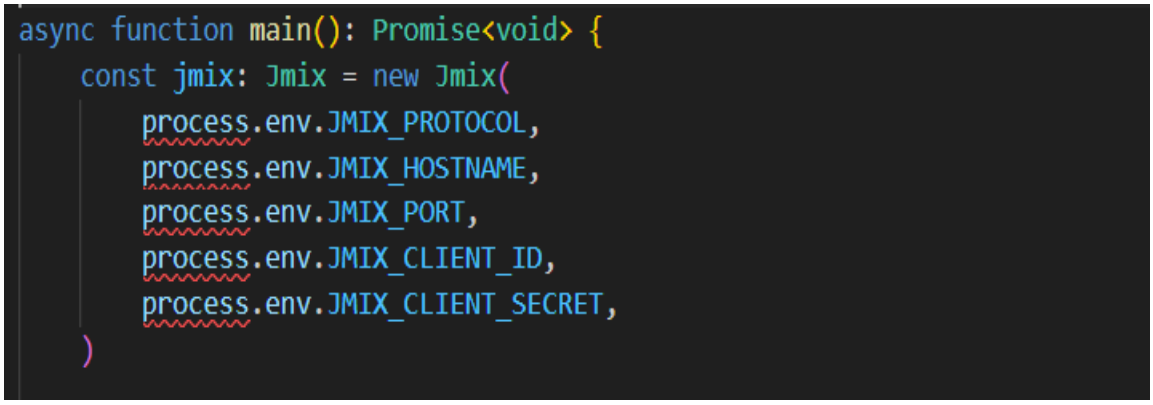


Figura 41. Envío de información al framework Jmix.

Luego como se muestra en la **Figura 42.** se procede a crear un servidor web http en Node, la creación y configuración de un servidor HTTP sirve para manejar solicitudes y respuestas. Este servidor puede expandirse para manejar diferentes rutas, implementar autenticación, servir archivos. Es la base sobre la cual se puede construir una aplicación web completa.

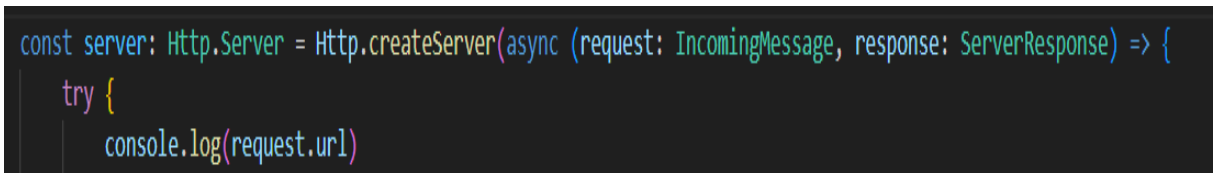


Figura 42. Implementación de un Servidor web http.

Como se muestra en la **Figura 43.** DotEnv es un paquete que se utiliza para cargar variables de entorno desde un archivo. /env que es una librería propia donde se crea una clase para guardar diferentes tipos de datos como configuraciones que pueden variar entre diferentes entornos (desarrollo, pruebas, producción), como claves API, URLs de bases de datos, puertos.

```
import DotEnv from 'dotenv'
DotEnv.config()

declare global {
    namespace NodeJS {
        interface ProcessEnv {
            NODE_ENV: 'development' | 'production'
            APP_PORT: number
            JMIX_PROTOCOL: string
            JMIX_HOSTNAME: string
            JMIX_PORT: string
            JMIX_CLIENT_ID: string
            JMIX_CLIENT_SECRET: string
        }
    }
}

export {}
```

Figura 43. Importación de paquete DotEnv.

Como se muestra en la **Figura 44.** En esta parte se define la ruta por la cual Node envía la información al servidor y comprueba que se encuentra funcionando correctamente. De igual manera empieza ejecutar el comando jmix.read, en donde empieza a realizar la lectura del manómetro y almacenar el resultado.

```
if(request.url === '/api/health') {
    response.writeHead(200, {'Content-Type': 'application/json'})
    response.write('ok')
} else {
    const allMeasurements: any[] = []

    const body: string = await getBody(request)
    const jmixManometers: any[] = await jmix.read('Manometer')
```

Figura 44. Definición de la ruta donde se enviará la información.

Luego, como se muestra en la **Figura 45.** se realiza la asignación de la medición desde el objeto manometer, para luego añadir la nueva medición. Para luego poder subir la información a la nube.

```
                const newMeasurement: any = await jmix.create('Measurement', {
                    manometer: {
                        id: matches[0].id
                    },
                    value: manometer['value']
                })
                allMeasurements.push(newMeasurement)
            }

            response.writeHead(200, {'Content-Type': 'application/json'})
            response.write(JSON.stringify(allMeasurements))
        }
    } catch(error) {
        console.error(error)
        response.statusCode = 500
    }
    response.end()
})

console.log(`http://127.0.0.1:${process.env.APP_PORT}`)
server.listen(process.env.APP_PORT)
```

Figura 45. Envío de información al servidor.

4.5 Etapa de visualización de datos

4.5.1 Implementación del aplicativo web

Para facilitar la visualización de datos se realiza la creación de un aplicativo web, en este proyecto como sé muestra en la **Figura 46.** se utilizará Vaadin que es un framework de desarrollo web que se utiliza para crear aplicaciones web con interfaces de usuario UI, Vaadin que permiten a los desarrolladores construir la interfaz de usuario utilizando Java, lo que facilita la integración y el manejo de la lógica de negocio y la presentación en un solo lenguaje. el lenguaje que se usará para realizar dicho aplicativo será JavaScript.

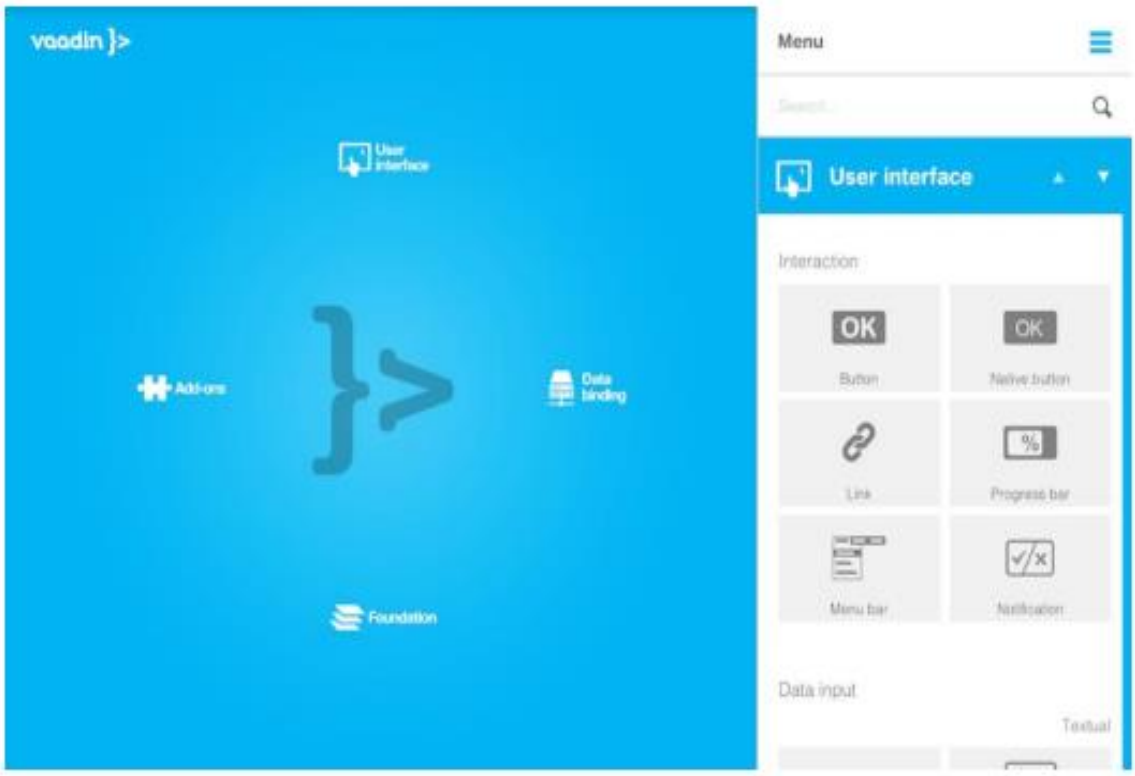


Figura 46. Marco de trabajo de Vaadin (Guachamin et al ., 2017).

Empezamos creando el código para poner funcional la base datos en Visual Studio Code, como se muestra en la **Figura 47.** Las 3 tablas donde se guardarán los datos son manometer.java, Measurement.java y user.java. tienen la extensión .java porque son fuentes escritas en el lenguaje de programación java.

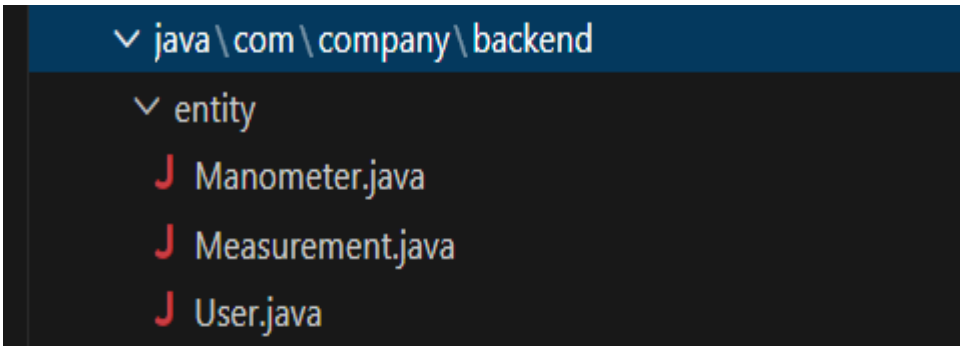


Figura 47. Base de datos de la aplicación web Elaboración propia.

Luego se procede a definir cada columna mediante el código que es de una clase java, la cual define una entidad llamada Manometer para ser utilizada en un contexto de JPA y Jmix. Como se muestra en la **Figura 48.** esta clase se utiliza para mapear la entidad Manometer a una tabla en la base de datos y gestionar sus propiedades como la Id, la versión, el nombre y su máximo valor.

```java
@JmixEntity
@Table(name = "MANOMETER")
@Entity
public class Manometer {
    @JmixGeneratedValue
    @Column(name = "ID", nullable = false)
    @Id
    private UUID id;

    @Column(name = "VERSION", nullable = false)
    @Version
    private Integer version;

    @InstanceName
    @Column(name = "NAME", nullable = false)
    protected String name;

    @Column(name = "MAX_VALUE", nullable = false)
    protected Double maxValue;

    @Column(name = "MIN_VALUE", nullable = false)
    protected Double minValue;
```

Figura 48. Definición de columnas mediante código java.

Como se muestra en la **Figura 49.** en java existe una librería que se llama liquibase, la cual convierte el código java de las columnas en XML, la razón es porque los navegadores web no entienden ni ejecutan directamente el lenguaje Java. Los navegadores están diseñados para interpretar y ejecutar HTML, CSS y JavaScript.

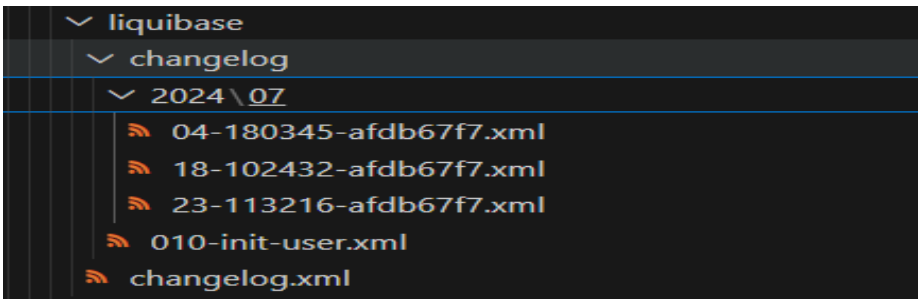


Figura 49. Implementación de la Librería liquibase.

Luego de la creación de las 3 tablas y su conversión al formato XML, liquibase lo analiza y genera un archivo en formato SQL para posteriormente enviarlo a la base de datos. Como se muestra en la **Figura 50.** Existen muchas tablas, pero solo se creó mediante programación las 3 principales (manometer, measurement y user), las demás fueron creadas por el framework.

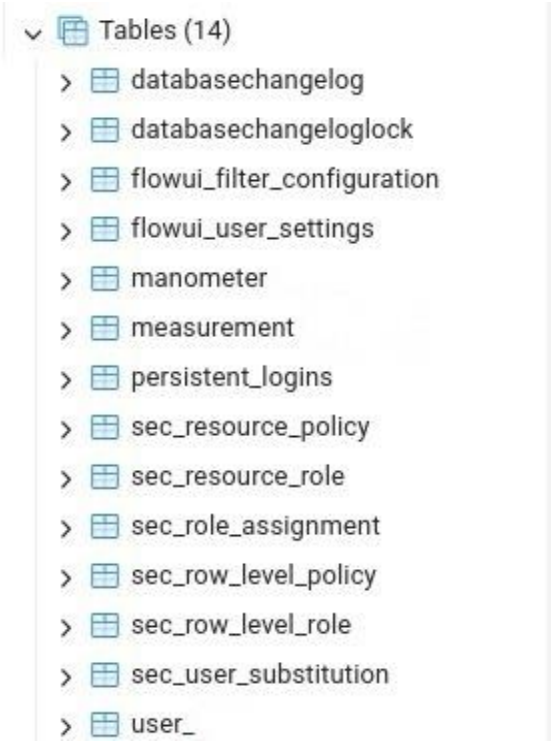


Figura 50. Base de datos de las tablas del manometer, measurement y el user.

Luego se procede a crear la vista de inicio de nuestra página web, al momento de crear se utiliza la librería Vaadin, que al igual que liquibase transformaba archivo formato java a formato XML o SQL, Vaadin se encargará de transformar archivos java en formato HTML, CSS y JavaScript, con la intención es permitir que esta librería genere automáticamente la UI y evitar realizar tanta programación, en la **Figura 51.** Se muestra el código en XML que realizará la configuración y estructura de la vista de inicio de sesión de nuestro aplicativo web utilizando el framework Jmix. Este código permite la configuración externa y establece un diseño centrado para el formulario del login, con opciones configurables como "Recordarme" y gestión de mensajes de error.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<view xmlns="http://jmix.io/schema/flowui/view"
      focusComponent="login"
      title="msg://LoginView.title">
    <layout justifyContent="CENTER" alignItems="CENTER" classNames="jmix-login-main-layout">
        <loginForm id="login"
                   rememberMeVisible="true"
                   forgotPasswordButtonVisible="false">
            <form title="msg://loginForm.headerTitle"
                  username="msg://loginForm.username"
                  password="msg://loginForm.password"
                  rememberMe="msg://loginForm.rememberMe"
                  submit="msg://loginForm.submit"
                  forgotPassword="msg://loginForm.forgotPassword"/>
            <errorMessage title="msg://loginForm.errorTitle"
                          message="msg://loginForm.badCredentials"
                          username="msg://loginForm.errorUsername"
                          password="msg://loginForm.errorPassword"/>
        </loginForm>
    </layout>
</view>
```

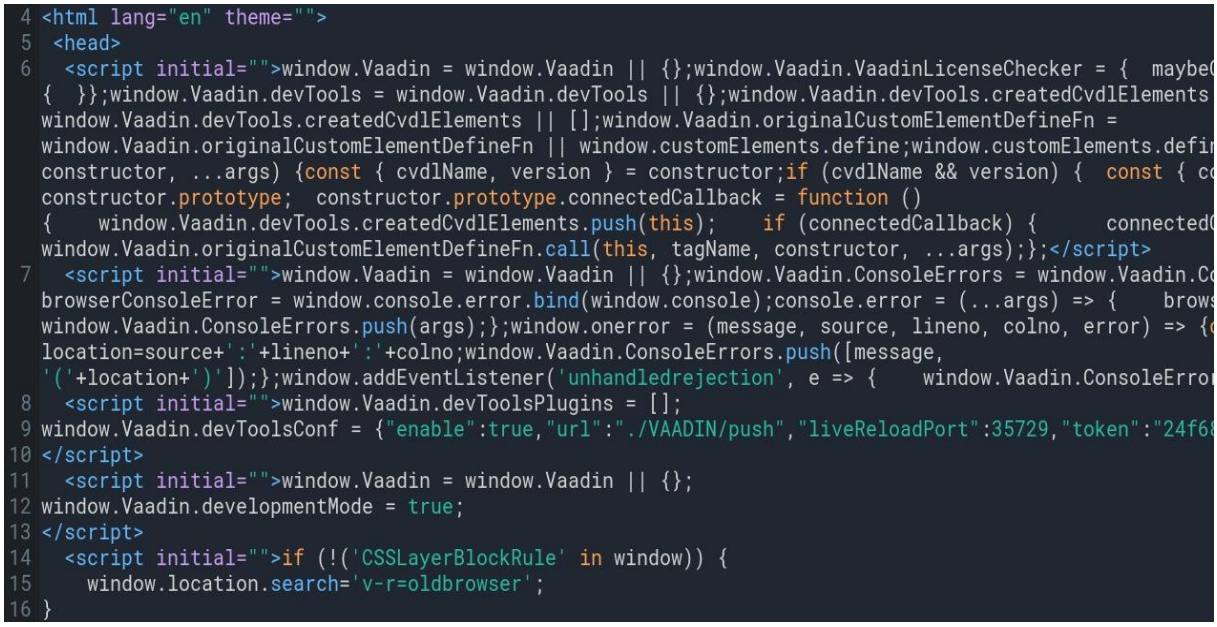
Figura 51. Código Java que genera la vista de inicio de nuestra página web.

El código que se encuentra en la **Figura 51.** es analizado por Vaadin para luego generar el Cascading Style Sheets o mejor conocido como CSS, como se muestra en la **Figura 52.**

```
2116   <style data-file-path="VAADIN/themes/backend/styles.css">/* Define your styles here */
2117 /* Predefined styles for the MainView */
2118
2119 .jmix-main-view-header {
2120   box-sizing: border-box;
2121   display: flex;
2122   height: var(--lumo-size-xl);
2123   align-items: center;
2124   width: 100%;
2125 }
2126
2127 .jmix-main-view-drawer-toggle {
2128   color: var(--lumo-secondary-text-color);
2129 }
2130
2131 .jmix-main-view-title {
2132   margin: 0;
2133   font-size: var(--lumo-font-size-l);
2134 }
```

Figura 52. CSS generado por Vaadin del login de la página web.

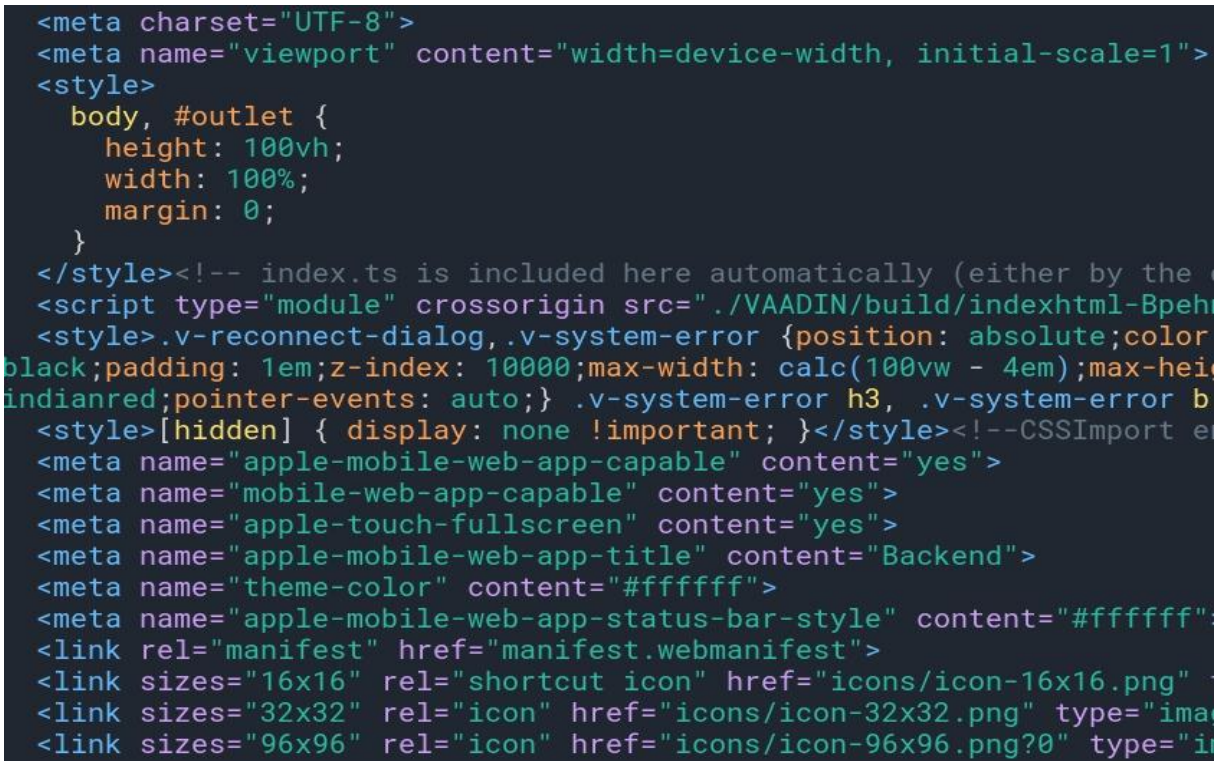
En la **Figura 53.** Se muestra el JavaScript generado por Vaadin del login de nuestra página web.



```
<html lang="en" theme="">
 <head>
  <script initial="">window.Vaadin = window.Vaadin || {};window.Vaadin.VaadinLicenseChecker = {  maybe(
 {  }};window.Vaadin.devTools = window.Vaadin.devTools || {};window.Vaadin.devTools.createdCvdlElements
 window.Vaadin.devTools.createdCvdlElements || [];window.Vaadin.originalCustomElementDefineFn =
 window.Vaadin.originalCustomElementDefineFn || window.customElements.define;window.customElements.defi
 constructor, ...args) {const { cvdlName, version } = constructor;if (cvdlName && version) {  const { c
 constructor.prototype;  constructor.prototype.connectedCallback = function ()
 {    window.Vaadin.devTools.createdCvdlElements.push(this);    if (connectedCallback) {      connected
 window.Vaadin.originalCustomElementDefineFn.call(this, tagName, constructor, ...args);};</script>
  <script initial="">window.Vaadin = window.Vaadin || {};window.Vaadin.ConsoleErrors = window.Vaadin.C
 browserConsoleError = window.console.error.bind(window.console);console.error = (...args) => {    brows
 window.Vaadin.ConsoleErrors.push(args);};window.onerror = (message, source, lineno, colno, error) => {
 location=source+':'+lineno+':'+colno;window.Vaadin.ConsoleErrors.push([message,
 '('+location+')']);};window.addEventListener('unhandledrejection', e => {    window.Vaadin.ConsoleErro
  <script initial="">window.Vaadin.devToolsPlugins = [];
window.Vaadin.devToolsConf = {"enable":true,"url":"./VAADIN/push","liveReloadPort":35729,"token":"24f6
</script>
  <script initial="">window.Vaadin = window.Vaadin || {};
window.Vaadin.developmentMode = true;
</script>
  <script initial="">if (!('CSSLayerBlockRule' in window)) {
    window.location.search='v-r=oldbrowser';
}
```

Figura 53. JavaScript generado por Vaadin del login de la página web.

En la **Figura 54.** Se muestra el HTML generado por Vaadin del login de nuestra página web.



```
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <style>
   body, #outlet {
     height: 100vh;
     width: 100%;
     margin: 0;
   }
 </style><!-- index.ts is included here automatically (either by the
 <script type="module" crossorigin src="./VAADIN/build/indexhtml-Bpeh
 <style>.v-reconnect-dialog,.v-system-error {position: absolute;color
black;padding: 1em;z-index: 10000;max-width: calc(100vw - 4em);max-hei
indianred;pointer-events: auto;} .v-system-error h3, .v-system-error b
 <style>[hidden] { display: none !important; }</style><!--CSSImport e
 <meta name="apple-mobile-web-app-capable" content="yes">
 <meta name="mobile-web-app-capable" content="yes">
 <meta name="apple-touch-fullscreen" content="yes">
 <meta name="apple-mobile-web-app-title" content="Backend">
 <meta name="theme-color" content="#ffffff">
 <meta name="apple-mobile-web-app-status-bar-style" content="#ffffff"
 <link rel="manifest" href="manifest.webmanifest">
 <link sizes="16x16" rel="shortcut icon" href="icons/icon-16x16.png"
 <link sizes="32x32" rel="icon" href="icons/icon-32x32.png" type="ima
 <link sizes="96x96" rel="icon" href="icons/icon-96x96.png?0" type="i
```

Figura 54. HTML generado por Vaadin del login de la página web.

Luego de la conversión que realiza Vaadin del código java que se escribió, se muestra la User Interface de la página web como se muestra en la **Figura 55.**

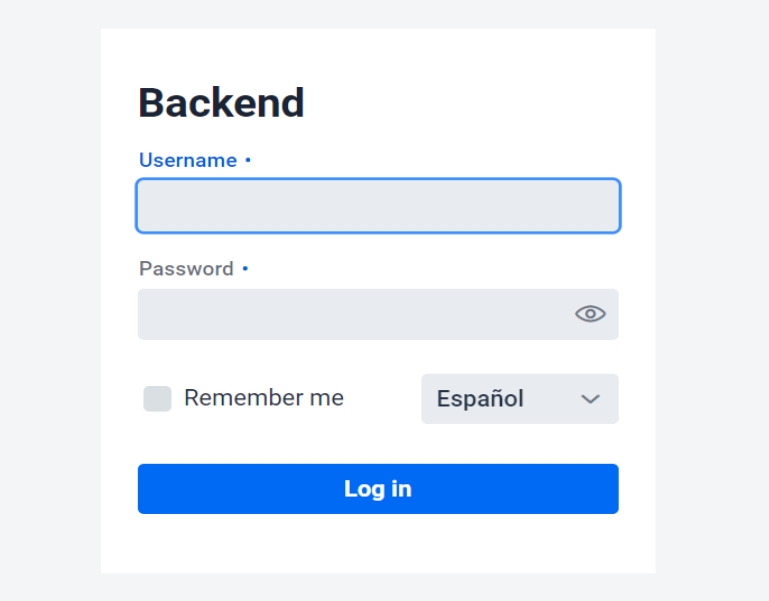


Figura 55. User de Interface de la página web.

Luego cuando el usuario desee ingresar o escribir el nombre del usuario, todas esas acciones, como dar click en Log in o inicio de sesión, se estará invocado al evento onLogin como se muestra en la **Figura 56.** Toda esta línea de código es la que permitirá ingresar al sistema en donde se encontrarán las mediciones del manómetro, también permitirá recuperar o recordar el usuario y contraseña y por último cambiar el idioma.

```
@Subscribe("login")
public void onLogin(final LoginEvent event) {
    try {
        loginViewSupport.authenticate(
                AuthDetails.of(event.getUsername(), event.getPassword())
                        .withLocale(login.getSelectedLocale())
                        .withRememberMe(login.isRememberMe())
        );
    } catch (final BadCredentialsException | DisabledException | LockedException | AccessDeniedException e) {
        log.warn("Login failed for user '{}': {}", event.getUsername(), e.toString());
        event.getSource().setError(true);
    }
```

Figura 56. Código que permite dar click al ingresar al sistema, cambiar o recuperar contraseña.

Luego se procede a crear las diferentes opciones o herramientas que va a tener la página web, como se muestra en la **Figura 57.** Se podrá observar los datos que genera el manómetro, la cantidad de oxígeno que está consumiendo y la cantidad de oxígeno que aún mantiene el tanque, el personal médico o usuarios que tendrán acceso a la página web, las mediciones que han sido guardadas en anteriores ocasiones y lo más importante se podrá tener monitoreado de manera remota mediante un dashboard las mediciones del tanque de manera real.

Figura 57. Herramientas de la página web.

En gran parte Vaadin ayuda a crear las diferentes herramientas que va a tener la página web, para que puedan enviar datos, de tal manera que se puedan observar en los registros. Como se muestra en la **Figura 58.** La clase java se encuentra vacía esto se debe porque Vaadin ya sabe cómo funciona una lista y como cargarla. La parte del código @ViewDescriptor("manometer-list-view.xml") indica que la clase a la que se aplica tiene asociado un descriptor de vista, que en este caso es un archivo.

```java
package com.company.backend.view.manometer;

import com.company.backend.entity.Manometer;
import com.company.backend.view.main.MainView;
import com.vaadin.flow.router.Route;
import io.jmix.flowui.view.*;


@Route(value = "manometers", layout = MainView.class)
@ViewController("Manometer.list")
@ViewDescriptor("manometer-list-view.xml")
@LookupComponent("manometersDataGrid")
@DialogMode(width = "64em")
public class ManometerListView extends StandardListView<Manometer> {
}
```

Figura 58. Clase Java de la lista del manómetro.

Los datos que se van a cargar en la página web se definen mediante una colección de la clase manómetro. Como se muestra en la **Figura 59.** este código le permite al sistema internamente cargar con la base y cargar los manómetros.

```
<data>
    <collection id="manometersDc"
                class="com.company.backend.entity.Manometer">
        <fetchPlan extends="_base">
            <property name="measurements" fetchPlan="_base"/>
        </fetchPlan>
        <loader id="manometersDl" readOnly="true">
            <query>
                <![CDATA[select e from Manometer e]]>
            </query>
        </loader>
    </collection>
</data>
```

Figura 59. Datos que se van a cargar a la página web.

En la **Figura 60.** Se observa que se crea un dataGrid o tabla donde se van a mostrar todos los manómetros, en esta tabla se le otorga la colección que se definió en la **Figura 59.** básicamente Vaadin sabe cómo funciona el dataGrid y a la misma vez sabe que si se ingresa al container, este automáticamente le muestra la lista de todos los manómetros.

```
<dataGrid id="manometersDataGrid"
          width="100%"
          minHeight="20em"
          dataContainer="manometersDc"
          columnReorderingAllowed="true">
```

Figura 60. DataGrid o tabla de los manómetros.

Como se muestra en la **Figura 61.** en la sección columna del DataGrid se debe definir que columnas se requieren mostrar, porque el manómetro puede tener algunas columnas, pero se debe definir que columnas se requiere mostrar, en este caso se muestra ´max value´, ´minValue´ y ´createdDate´.

```
<columns resizable="true">
    <column property="name"/>
    <column property="maxValue"/>
    <column property="minValue"/>
    <column property="createdDate"/>
</columns>
```

Figura 61. Definición de columnas que se van a mostrar.

En la **Figura 62.** se observan los valores del manómetro tanto derecho como izquierda.

Nombre	Valor máximo	Valor mínimo	Created date
left	15	0	07/07/2024 1...
right	3,000	300	07/07/2024 1...

Figura 62. IU de las columnas de los manómetros que se van a mostrar.

Luego se procede a generar un filtro de búsqueda, para que el usuario cuando desee una medición de alguna fecha y hora especifica, no deba buscar medición por medición, al contrario, se realiza mediante un filtro de búsqueda, el cual se genera con el código que se muestra en la **Figura 63.**

```
<genericFilter id="genericFilter"
               dataLoader="manometersDl">
    <properties include=".*"/>
```

Figura 63. Código para el filtro de búsqueda de mediciones.

Como se muestra en la **Figura 64.** Se logra visualizar la Interface User del filtro de búsqueda.

Figura 64. IU del filtro de búsqueda.

Luego se procede con la creación de botones para crear, editar o eliminar que apuntan a alguna acción, como se muestra en el código de la **Figura 65.**

```
<hbox id="buttonsPanel" classNames="buttons-panel">
    <button id="createBtn" action="manometersDataGrid.create"/>
    <button id="editBtn" action="manometersDataGrid.edit"/>
    <button id="removeBtn" action="manometersDataGrid.remove"/>
    <simplePagination id="pagination" dataLoader="manometersDl"/>
```

Figura 65. Creación de botones para crear, editar o eliminar.

Como observamos en la **Figura 66.** estos botones pueden crear filtraciones, por ejemplo, mediciones que se realizaron en un día especifico y a una hora exacta, también se puede editar mediciones, por ejemplo, el valor máximo y mínimo que se desea tener en alguna medición y a la misma vez tener la opción de eliminar algunas filtraciones o mediciones.

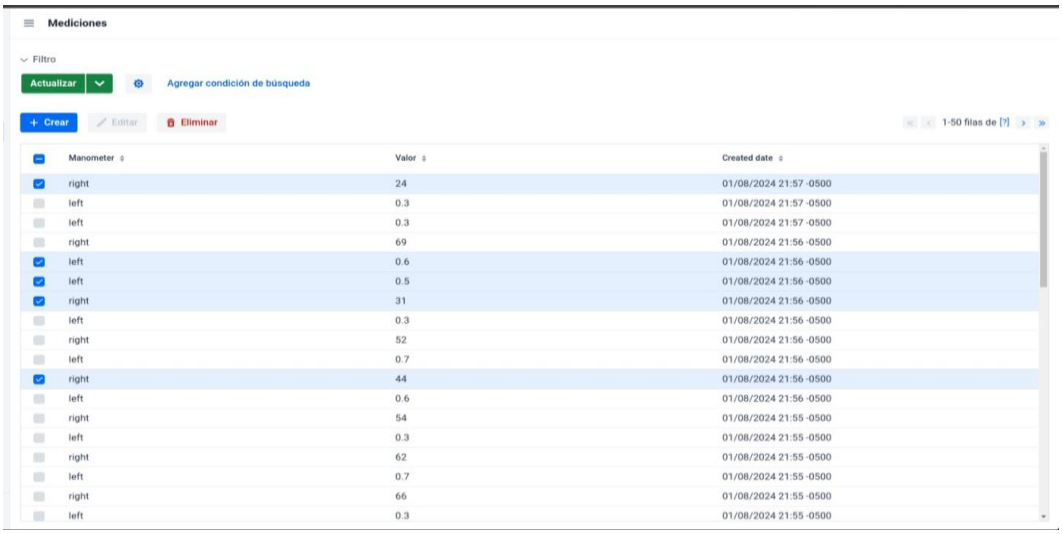


Figura 66. Funcionamiento de los botones de filtración.

Al igual como se creó las mediciones y los manómetros en la página web, para crear la parte del dashboard se debe realizar una programación en java como se muestra en la **Figura 67.** y en esa programación cargar un código XML.

Figura 67. Código java para la creación del dashboard.

Luego en la sección data del código XML, como se observa en la **Figura 68.** se procede a cargar los datos del manómetro tanto el derecho como el izquierdo.

code > backend > src > main > resources > com > company > backend > view > dashboard > dashboard-view.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<view xmlns="http://jmix.io/schema/flowui/view"
      xmlns:charts="http://jmix.io/schema/charts/ui"
      title="msg://dashboardView.title">
    <data>
        <collection id="measurementsRightDc"
                    class="com.company.backend.entity.Measurement">
            <fetchPlan extends="_base">
                <property name="manometer" fetchPlan="_base"/>
            </fetchPlan>
            <loader id="measurementsRightDl" readOnly="true">
                <query>
                    <![CDATA[select e from Measurement e where e.manometer.name = :name and e.createdDate >= :createdDate order by e.createdDate asc]]>
                </query>
            </loader>
        </collection>
        <collection id="measurementsLeftDc"
                    class="com.company.backend.entity.Measurement">
            <fetchPlan extends="_base">
                <property name="manometer" fetchPlan="_base"/>
            </fetchPlan>
            <loader id="measurementsLeftDl" readOnly="true">
                <query>
                    <![CDATA[select e from Measurement e where e.manometer.name = :name and e.createdDate >= :createdDate order by e.createdDate asc]]>
                </query>
            </loader>
        </collection>
    </data>
```

Figura 68. Código XML donde se cargan los datos del manómetro.

Luego para realizar el gráfico del dashboard, se procede a utilizar el componente Charts que es una herramienta que permite a los desarrolladores agregar gráficos interactivos y altamente personalizables a sus aplicaciones web. Como se muestra en el código de la **Figura 69.** Se utilizo este componente para definir los tooltip, los dataset y los axes dependiendo lo que se desea visualizar.

```xml
<charts:chart width="100%" height="100%" minHeight="50%" alignSelf="STRETCH">
    <charts:tooltip trigger="AXIS">
        <charts:axisPointer type="SHADOW"/>
    </charts:tooltip>
    <charts:legend/>

    <charts:dataSet>
        <charts:source dataContainer="measurementsRightDc" categoryField="createdDateTime" valueFields="value, manometer.maxValue, manometer.minValue"/>
    </charts:dataSet>

    <charts:xAxes>
        <charts:xAxis/>
    </charts:xAxes>
    <charts:yAxes>
        <charts:yAxis name="msg://rightManometer"/>
    </charts:yAxes>

    <charts:series>
        <charts:line name="msg://rightManometer" yAxisIndex="0"/>
        <charts:line name="msg://maxValue" yAxisIndex="0">
            <charts:itemStyle color="#910000"/>
        </charts:line>
        <charts:line name="msg://minValue" yAxisIndex="0">
            <charts:itemStyle color="#910000"/>
        </charts:line>
```

Figura 69. Código para la creación del dashboard.

5 RESULTADOS

En este capítulo se detallan los resultados obtenidos, posteriores al procesamiento y a la programación realizada. Al terminar este proceso se logrará observar de diferentes maneras las mediciones remotas que se han realizado al manómetro de un tanque de oxígeno y de la misma manera se logrará visualizar todas las mediciones en forma de gráfico en una página web.

5.1 Resultados del reconocimiento de las perillas del manómetro.

Existen diferentes maneras de comprobar el reconocimiento exitoso de las perillas, una manera es la que se visualiza en la **Figura 70.** Se puede comprobar como la cámara realiza el reconocimiento de las perillas insufladoras del manómetro y emite dos valores distintos, el primero valor significa que el manómetro está proporcionando una cantidad de 3.61 litros por minutos y la segunda cantidad que es 1525.85 litros, es la cantidad de oxígeno que mantiene almacenado el tanque. Esta visualización se la realiza ejecutando el código de Python en la raspberry desde el command prompt.

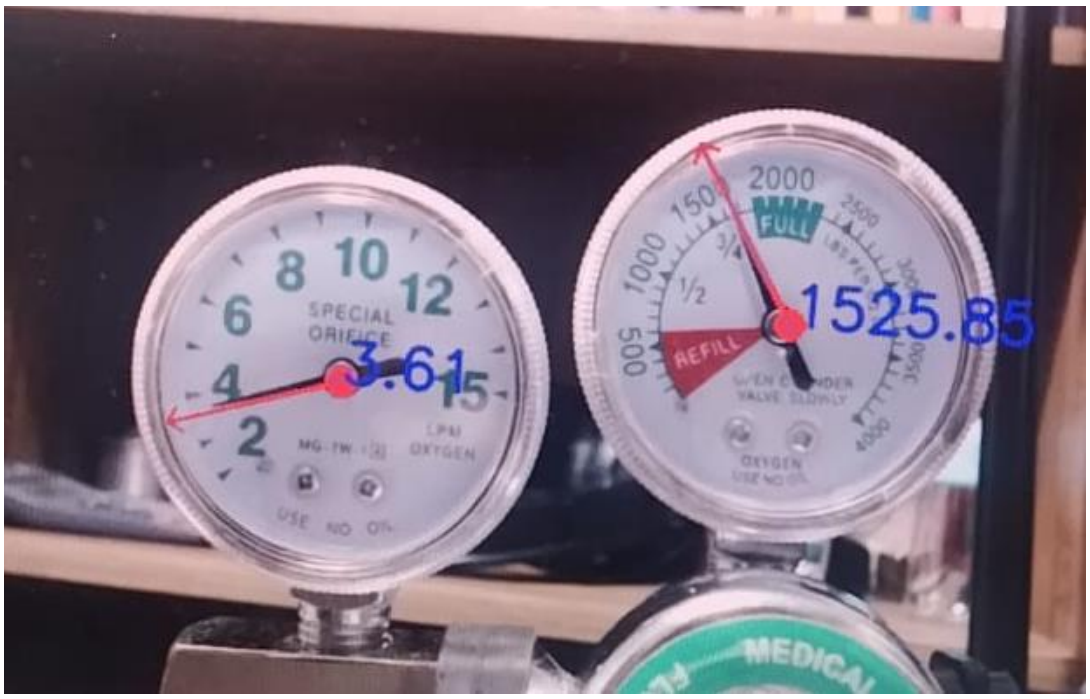


Figura 70. Reconocimiento de las parillas del manómetro.

5.2 Resultado de la página web

Otra manera de comprobar que la cámara se encuentre realizando con éxito el reconocimiento del manómetro es mediante la página web. Para acceder a la página web se debe ingresar al URL: http://manometermeasurer.eastus.cloudapp.azure.com/, luego se podrá visualizar la página de inicio como se muestra en la **Figura 71.** En donde ingresamos nuestro usuario y contraseña para acceder a las diferentes herramientas que maneja la página web. Una de las herramientas es comprobar que la cámara está reconociendo las perillas y eso se realiza ingresando a la sección de mediciones, como se muestra en la **Figura 72.** en donde se comprueba mediante los valores mostrados que se encuentran en esta sección, que efectivamente la cámara está realizando su trabajo de manera correcta. En la sección de mediciones también encontramos opciones que nos permite eliminar mediciones innecesarias, crear filtraciones de búsqueda, por ejemplo, buscar una medición especifica de un día exacto con un rango de horario definido y a la misma vez nos permite editar este tipo de filtraciones y si deseamos también se pueden eliminar.

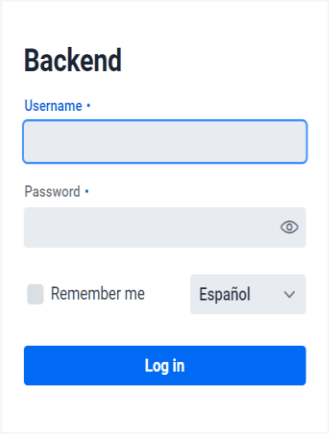


Figura 71. Página de inicio de nuestra página web.

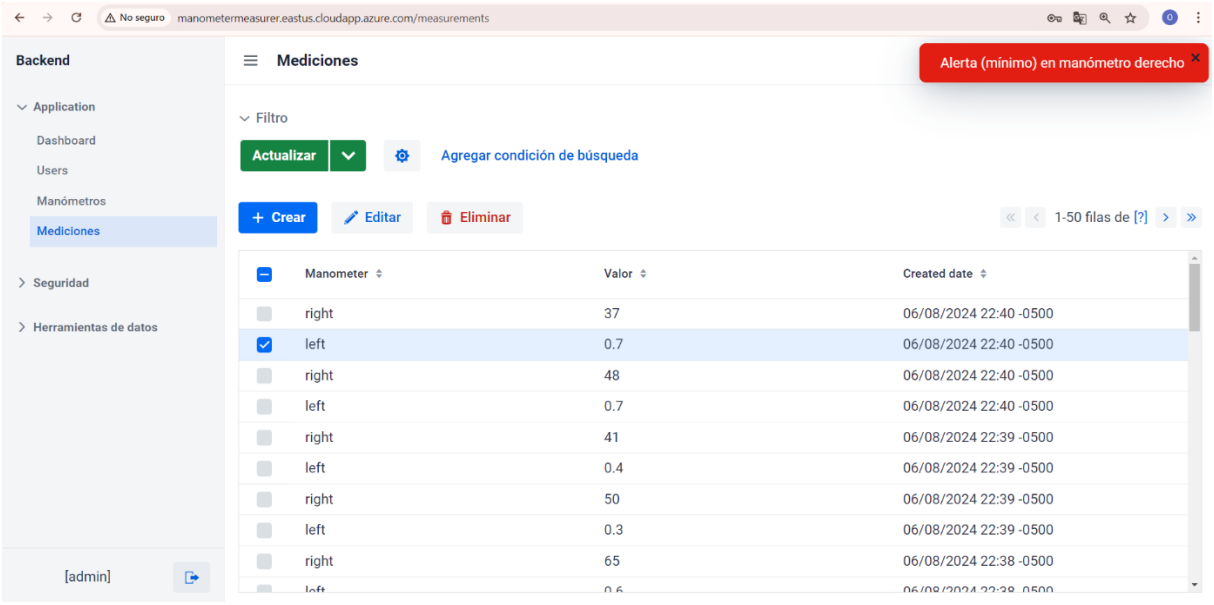


Figura 72. Sección de mediciones.

Otra manera de comprobar o visualizar los valores del manómetro es mediante gráficos, por esta razón en la página web se creó una sección que se llama Dashboard, como se muestra en la **Figura 73.** en donde se podrá visualizar mediante un polígono de frecuencia los valores que se encuentran en el manómetro, los cuales están siendo captados por la cámara web. Adicional se agregó una señal de alerta, la cual sirve para avisar al personal médico cuando el tanque de oxígeno este por debajo del umbral crítico.

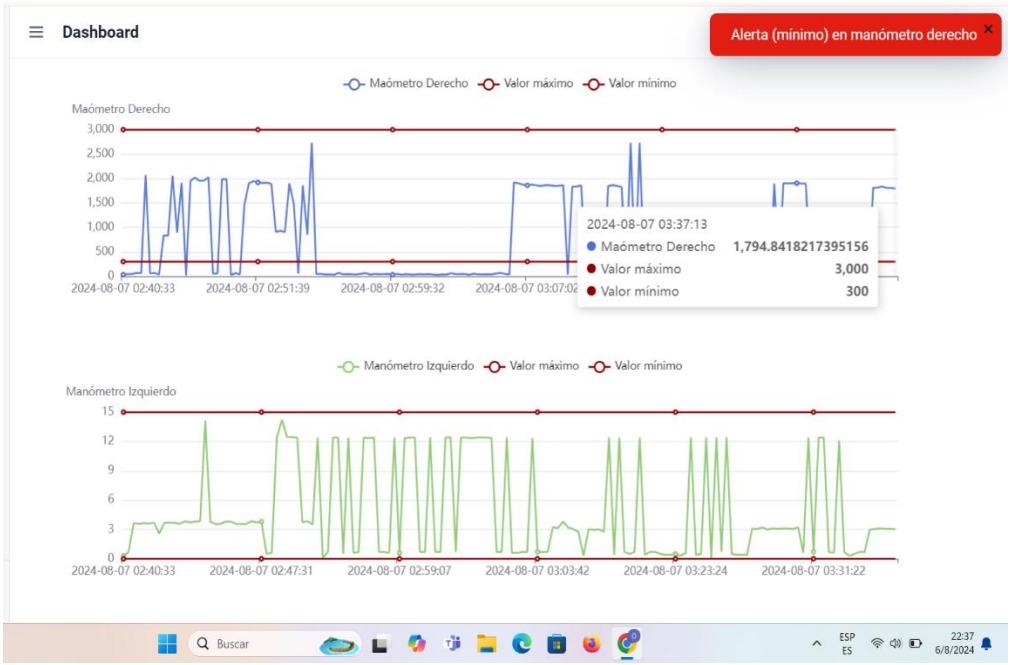


Figura 73. Dashboard de la página web. Elaboración propia.

En nuestra página web también se creó la sección de manómetros, en donde se encontrará los dos manómetros y como se muestra en la **Figura 74.** se podrá configurar los valores máximos y mínimos que se desea estandarizar, para cuando exista un valor fuera del rango programado, se emita una señal de alerta.

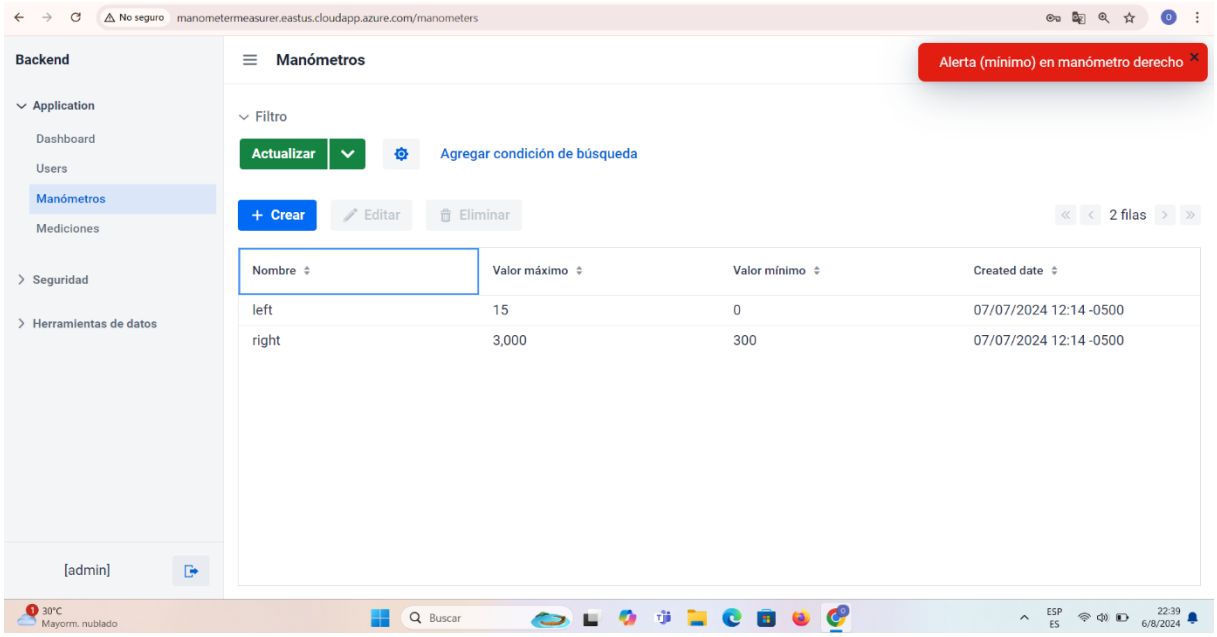


Figura 74. Sección de manómetros.

Como se muestra en la ***Figura 75.*** dentro de la página web también se encuentra la sección usuarios, en donde se podrá configurar los usuarios que tendrán acceso a la página web.

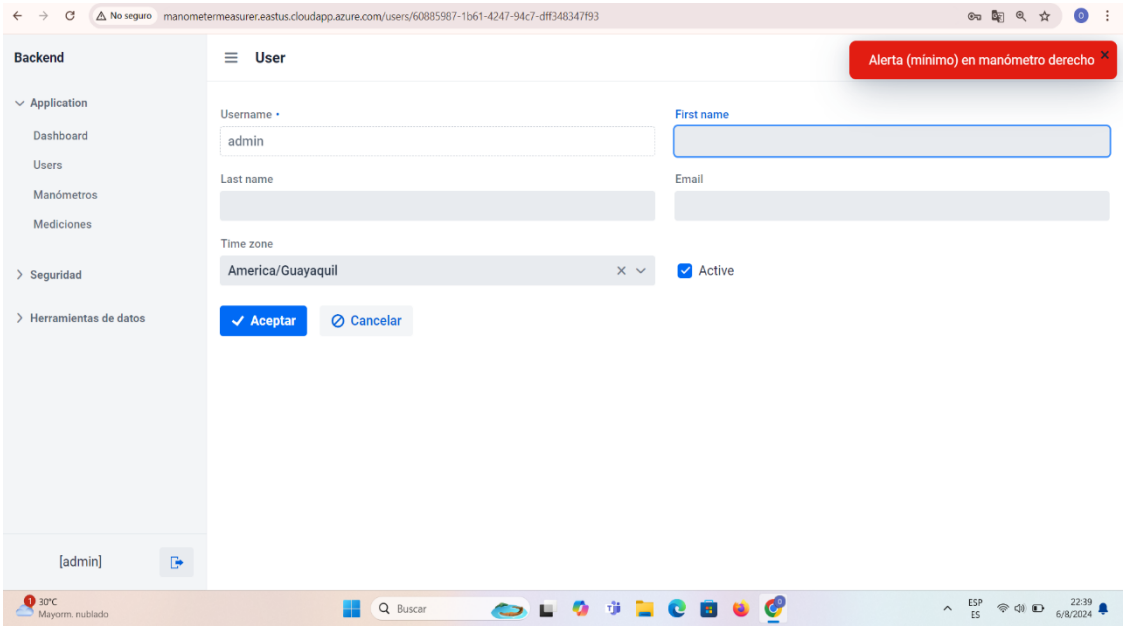


Figura 75. Sección de usuario de la página web.

Como se observa en la **Figura 76.** la lista de herramientas que tiene la página web, existe la sección que se llama Seguridad, en esta sección se puede restringir a los usuarios de ciertas herramientas, por ejemplo, se puede configurar que una persona pueda visualizar los valores del manómetro dentro de la sección “medición”, pero que tenga acceso restringido al dashboard y a la sección manómetros y a la misma vez que otra persona tenga acceso al dashboard pero que no pueda ingresar a la parte de manómetros.

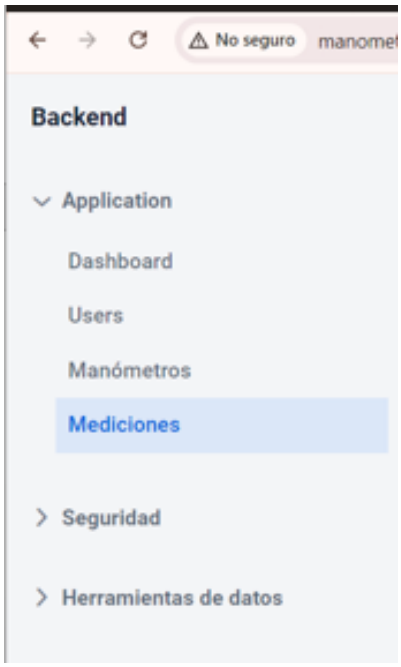


Figura 76. Sección de seguridad de la página web.

CRONOGRAMA

Las actividades del trabajo de titulación se realizaron en un tiempo aproximado de 12 semanas como se puede observar en la **Tabla 1.**

Actividades	SEMANAS											
	1	2	3	4	5	6	7	8	9	10	11	12
Compra de equipos y componentes.												
Investigación de funcionamiento de cada equipo.												
Configuración de la raspberry para el reconocimiento del manómetro con IA.												
Configuración de la cámara para la lectura de los manómetros												
Configuración del sistema Node red para el envío y recepción de datos												
Diseño de aplicación web en Azure												
Diseño del soporte para el raspberry pi4												
Implementación de un sistema de luminarias para una mejor lectura del manómetro												

Tabla 1. Cronograma de actividades.

PRESUPUESTO

A continuación, se puede observar en la **Tabla 2**. el presupuesto de todos los materiales que se utilizó en el proyecto, siendo el Raspberry Pi4 uno de los elementos con mayor valor económico.

Descripción	Cantidad	Costo unitario	Costo final
Raspberry Pi4	1	$120,00	$120,00
Cargador Raspberry	1	$10,00	$10,00
Micro SD	1	$20,00	$20,00
USB Camera	1	$60,00	$60,00
Manómetro de oxígeno	1	$110,00	$110,00
Pin conector hembra y macho	20	$1	$20,00
Soporte ajustable para raspberry	1	$40,00	$40,00
Sistema de luminarias	1	$15	$15
Total			$415,00

Tabla 2. Tabla de presupuesto.

CONCLUSIONES

- Para el reconocimiento de las diferentes posiciones de las perillas del manómetro se realizó diferentes reconocimientos tales como reconocer los círculos, los colores verdes y rojos del manómetro, para este reconocimiento se uso la librería OpenCV de Python.

- Los datos que se visualizó por la cámara se envían al sistema NODE RED, que los transmite a la nube para su visualización en una página web, permitiendo a los responsables del cuidado del paciente monitorear los niveles de oxígeno en tiempo real.

- Se programó una señal de alerta la cual es fundamental para garantizar la disponibilidad continua de oxígeno, facilitando la sustitución oportuna del tanque y mejorando la atención al paciente en situaciones de emergencia.

RECOMENDACIONES

- Se recomienda realizar pruebas exhaustivas y validaciones con diferentes niveles de oxígenos en el tanque para asegurar que el sistema funcione correctamente.

- Se recomienda implementar un plan de mantenimiento regular para la cámara y el hardware asociado para asegurar su correcto funcionamiento a largo plazo.

- Se estableció mecanismos de autenticación y autorización robustos para garantizar que solo el personal autorizado pueda acceder a los datos sensibles.

REFERENCIAS BIBLIOGRAFICAS

Abásolo, A. F. (2005). *Revista Oficial de la Sociedad andaluzia*. Obtenido de http://esperanzaquintero.es/documentos/SAFH%2080-85.pdf

Alonso, C. R. (11 de Febrero de 2021). *ÓRGANO INFORMATIVO DE LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO*. Obtenido de https://www.gaceta.unam.mx/la-escasez-de-oxigeno-por-fractura-en-la-distribucion/

Baquero, R. C. (Noviembre de 2020). *Universidad San Marco Repositorio*. Obtenido de https://repositorio.usam.ac.cr/xmlui/handle/11506/2119

Baz Alonso, A., Ferreira Artime, I., Álvarez Rodríguez, M., & García Baniello , R. (2011). *Universidad de Oviedo*. Obtenido de https://d1wqtxts1xzle7.cloudfront.net/34258261/dispostivos_moviles_y_su_clasificacion-libre.pdf?1405966120=&response-content-disposition=inline%3B+filename%3DDispositivos_moviles.pdf&Expires=1720042387&Signature=g69XTLbyDW7RbGeD4y9VtQOU4A4DsjL7NVjlGr6XAlu

Challenger Pérez, I., Díaz Ricardo , Y., & Becerra García , R. A. (02 de Abril-Junio de 2014). *Ciencias Holguín* . Obtenido de www.redalyc.org/articulo.oa?id=181531232001

Chávez , P., Zambrano Romero, O. E., & Toala Paz, A. X. (2009). *DSpace Espol*. Obtenido de https://www.dspace.espol.edu.ec/handle/123456789/8826

Chavez, E. D. (01 de 07 de 2022). *Centro de Recursos para el aprendizaje y la investigación*. Obtenido de https://repositorio.unitec.edu/xmlui/handle/123456789/12458

David, R. U. (2017). *Repositorio de la Universidad de ECCI*. Obtenido de https://repositorio.ecci.edu.co/handle/001/1209

Dominguez Gil, H. A. (Septiembre - octubre de 2005). Obtenido de https://www.redalyc.org/pdf/3659/365974238002.pdf

GEOVANNY, S. C. (Mayo de 2019). *Dspace Instituto Tecnológico Superior Vida Nueva*. Obtenido de http://dspace.istvidanueva.edu.ec/xmlui/bitstream/handle/123456789/91/44.0260-SUNTAXI-CANTU%C3%91A-OSCAR-GEOVANNY.pdf?sequence=1

Google maps. (2008).

google maps. (2018).

Guachamin Saguay, C. E., & Hurtado Guapulema, D. A. (Noviembre de 2017). *Repositorio de la Escuela Superior Politécnica de Chimborazo*. Obtenido de http://dspace.espoch.edu.ec/handle/123456789/9294

INFRA DE HONDURAS. (2020). Obtenido de https://www.infradehonduras.com.hn/oxido-nitroso/

Jolles, J. W. (23 de Junio de 2021). *British Ecological Society*. Obtenido de https://doi.org/10.1111/2041-210X.13652

Miguel, M. P. (1 de Diciembre de 2021). *Universidad de Jaén*. Obtenido de https://crea.ujaen.es/handle/10953.1/15335

Montalvo López, M. F. (octubre de 2010). *Repositorio Institucional de la Universidad Politécnica Salesiana*. Obtenido de https://dspace.ups.edu.ec/handle/123456789/2205

Moreno García, F. H. (09 de Septiembre de 2014). *Repositorio de la Universidad Pontificia Bolivariana*. Obtenido de http://hdl.handle.net/20.500.11912/1664

PASCUAL, L. R. (s.f.). *Oxígeno en el tratamiento de la insuficiencia*. Obtenido de neumologiaysalud.es

Pérez López , C., & Samá Monsonís , A. (Julio de 2013). *Universitat Politécnica de Catalunya*. Obtenido de https://upcommons.upc.edu/handle/2099.1/19142

Pérez López, C., & Samá Monsonís, A. (Julio de 2023). *Universitat Politecnica de Catalunya*. Obtenido de https://upcommons.upc.edu/handle/2099.1/19142

Quintana Lopez, M., Flores Albino, J. M., Landassuri Moreno, V. M., Lazcano Salas, S., Morales Escobar, S. J., & Orozco Aguirre, R. H. (2015). Comparación de Algoritmos de Agrupamiento. En C. A. Computacional, *Avances en Sistemas y Computación* (pág. 28). Toluca.

Rala, M. A. (Julio de 2021). *Archivo Digital Universidad Politécnica de Madrid*. Obtenido de Aplicación de aprendizaje profundo: clasificación de imágenes médicas basado en servicios en la nube: https://oa.upm.es/69406/

Romero, L. M. (Octubre-diciembre de 2023). *Cuadernos Técnicos Universitarios de la DGTIC*. Obtenido de https://cuadernos.tic.unam.mx/index.php/cua/article/view/24

Universo, E. (6 de Mayo de 2021). Lo que debes saber para comprar o llenar un tanque de oxígeno medicinal. *Quienes padecen de covid-19 pueden necesitar de un tanque de oxígeno.*

Vlaeminch, R. C. (04 de Marzo de 2016). *Universidad de La Laguna*. Obtenido de https://riull.ull.es/xmlui/bitstream/handle/915/1930/Deep+Learning+para+reconocimiento+de+imagenes+en+Raspberry+Pi+2.pdf;jsessionid=115E4868E142CDC544789574501EADCE?sequence=1

Yarzábal, T., Alzate, I., & Mussini, P. (Octubre de 2018). *Revista de Investigación Clínica y Biomédica*. Obtenido de https://revistasaludmilitar.uy/ojs/index.php/Rsm/article/view/6