Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

# DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB EN TIEMPO REAL

**Kelvin René García Chinchilla**
Asesorado por el Ing. Guillermo Antonio Puente Romero

Guatemala, octubre de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


FACULTAD DE INGENIERÍA

# DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB EN TIEMPO REAL

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

**KELVIN RENÉ GARCÍA CHINCHILLA**
ASESORADO POR EL ING. GUILLERMO ANTONIO PUENTE ROMERO

AL CONFERÍRSELE EL TÍTULO DE

**INGENIERO ELECTRÓNICO**

GUATEMALA, OCTUBRE DE 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA


## NÓMINA DE JUNTA DIRECTIVA

| | |
|---|---|
| DECANA | Inga. Aurelia Anabela Cordova Estrada |
| VOCAL I | Ing. José Francisco Gómez Rivera |
| VOCAL II | Ing. Mario Renato Escobedo Martínez |
| VOCAL III | Ing. José Milton de León Bran |
| VOCAL IV | Br. Kevin Vladimir Cruz Lorente |
| VOCAL V | Br. Fernando José Paz González |
| SECRETARIO | Ing. Hugo Humberto Rivera Pérez |

## TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

| | |
|---|---|
| DECANA | Inga. Aurelia Anabela Cordova Estrada |
| EXAMINADOR | Ing. Christian Antonio Orellana López |
| EXAMINADOR | Ing. Walter Jacobo Galicia García |
| EXAMINADOR | Ing. Carlos Alberto Fernando Navarro Fuentes |
| SECRETARIO | Ing. Hugo Humberto Rivera Pérez |

# HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

**DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB EN UN TIEMPO REAL**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 16 de marzo de 2021

**Kelvin René García Chinchilla**

Guatemala, 16 de septiembre de 2021.

Ing. Julio Solares Peñate
Coordinador de Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Ingeniero Solares:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **"DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB ENTIEMPO REAL"**, desarrollado por el estudiante Kelvin René García Chinchilla carné No. 2013-14118, ya que considero que cumple con los requisitos establecidos, por lo que el autor y mi persona somos responsables del contenido y conclusiones del mismo.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

Ing. Guillermo Antonio Puente Romero
ASESOR
Colegiado 5898

Guillermo A. Puente R.
INGENIERO ELECTRÓNICO
COL # 5898

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 23 de septiembre de 2021

**Señor director**
**Armando Alonso Rivera Carrillo**
**Escuela de Ingeniería Mecánica Eléctrica**
**Facultad de Ingeniería, USAC**

Estimado Señor director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB ENTIEMPO REAL,** desarrollado por el estudiante **Kelvin René García Chinchilla**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

**ID Y ENSEÑAD A TODOS**

**Ing. Julio César Solares Peñate**
**Coordinador de Electrónica**

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

**REF. EIME 152 2021.**

**El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; KELVIN RENÉ GARCÍA CHINCHILLA, titulado: DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB EN TIEMPO REAL, procede a la autorización del mismo.**

**Ing. Armando Alonso Rivera Carrillo**

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

**GUATEMALA, 19 DE OCTUBRE 2,021.**

**Decanato**
**Facultad de Ingeniería**
**24189101 – 24189102**
**secretariadecanato@ingenieria.usac.edu.gt**

DTG. 582-2021

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE BAJO COSTO PARA EL MONITOREO DEL NIVEL DE AGUA POTABLE EN CISTERNAS A TRAVÉS DE UNA APLICACIÓN WEB EN TIEMPO REAL,** presentado por el estudiante universitario: **Kelvin René García Chinchilla,** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Inga. Anabela Cordova Estrada
Decana

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala, octubre de 2021

AACE/cc

## ACTO QUE DEDICO A:

| | |
|---|---|
| **Dios** | Por permitirme el privilegio de culminar esta etapa de mi vida, por ser mi guía en todo lo que realizo y darme las capacidades necesarias para desarrollar cada meta propuesta. |
| **Mis padres** | Mario García y Nilda Chinchilla, por motivarme y estar allí en cada paso del proceso, por su esfuerzo constante, ejemplo a lo largo de mi vida y por sus oraciones. |
| **Mi hermano** | David García, por ser un importante apoyo en mi carrera, entre otras cosas. |
| **Mis abuelos** | Nohelia Palma y Guillermo Chinchilla, por su apoyo, motivación y oraciones que fortalecieron esta etapa de mi vida. |
| **Mis amigos** | Manuel Fernández, Fabiola España y Miguel Tavico por el apoyo, lecciones, ayuda y momentos inolvidables durante esta etapa |
| **Gabriela Roldán** | Por motivarme, apoyarme y estar allí en los momentos necesarios e inolvidables durante esta etapa de mi vida. |

# AGRADECIMIENTOS A:


| | |
|---|---|
| **Universidad de San Carlos de Guatemala** | Por ser mi casa de estudios y por brindarme las herramientas para convertirme en un profesional. |
| **Facultad de Ingeniería** | Por ser el lugar en donde he desarrollado mis conocimientos. |
| **Ing. Guillermo Puente** | Por su guía en el proceso, y buen ejemplo de manera profesional y académica. |

# ÍNDICE GENERAL

# ÍNDICE DE ILUSTRACIONES

## FIGURAS

## TABLAS

# LISTA DE SÍMBOLOS

| Símbolo | Significado |
|---|---|
| **cm** | Centímetro |
| **Hz** | Hercio |
| **kHz** | Kilohercio |
| **m** | Metro |
| **µs** | Microsegundo |
| **s** | Segundo |
| **Vin** | Voltaje de entrada |
| **v** | Voltaje |

# GLOSARIO

| | |
|---|---|
| **Cliente** | Elemente de una red capaz de realizar peticiones a un servidor y realizar funciones específicas. |
| **CSS** | Lenguaje de hojas de estilo en cascada. |
| **Echo** | Onda sonora reflejada. |
| ***Framework*** | Estructura establecida para el desarrollo de software. |
| **Hardware** | Partes físicas que componen un sistema informático. |
| **HTML** | Lenguaje de marcas de hipertexto. |
| **HTTP** | Protocolo de transferencia de hipertexto. |
| **IP** | Dirección establecida por un conjunto de cuatro números a los equipos que conforman una red. |
| **Jinja2** | Motor de plantillas desarrollado en python. |
| **Micro-USB** | Conector USB de pequeñas dimensiones. |
| **POST** | Prueba de encendido automático. |

| | |
|---|---|
| **Red** | Conjunto de dispositivos interconectados entre sí que intercambian o comparten información. |
| **RX** | Parte receptora de un sistema electrónico. |
| **Servidor** | Equipo físico integrado a una red con la funcionalidad de atender peticiones de un cliente. |
| **Software** | Parte lógica que compone las funciones de un sistema informático. |
| **SQL** | Lenguaje de alto nivel orientado a registros. |
| **Sqlite** | Sistema de gestión de bases de datos contenida. |
| ***Trigger*** | Onda sonora emitida por medio de un pulso. |
| **TX** | Parte transmisora de un sistema electrónico. |
| **Voltaje** | Potencial eléctrico. |
| **Wifi** | Tecnología de conexión inalámbrica. |

# RESUMEN

En el presente trabajo de graduación, que consta de cuatro capítulos, se desarrolla el diseño y se documenta la implementación de un dispositivo de bajo costo para el monitoreo de nivel de agua potable en cisternas a través de una aplicación web en tiempo real.

En el capítulo 1 se presentan los distintos tipos de medidores de nivel de agua potable, sus características principales y su clasificación por método el cual puede ser directo e indirecto.

En el capítulo 2 se presenta cada uno de los dispositivos por utilizar, las características de uso, métodos de implementación y funcionamiento. Se analiza el diseño de hardware propuesto para el desarrollo del dispositivo, así como las conexiones por realizar para la integración del dispositivo.

En el capítulo 3 se presenta el diseño del código del software, por medio de etapas que clasifican las funcionalidades principales. Se define que el software consta de dos partes de las cuales una es desarrollada en la tarjeta y otra en el servidor.

En el capítulo 4 se documenta la implementación del dispositivo y el diseño final integrando el diseño en software y hardware, se realizan pruebas de funcionamiento del dispositivo.

En el capítulo 5 se documentan los costos y horas de trabajo utilizados para la implementación del dispositivo.

# OBJETIVOS

## General

Diseñar e implementar un dispositivo de bajo costo para el monitoreo del nivel de agua potable en cisternas a través de una aplicación web en tiempo real.

## Específicos

1. Presentar los distintos tipos de medidores de nivel de agua potable y su clasificación por método.

2. Presentar el diseño y configuración del hardware del dispositivo de monitoreo de nivel del agua potable.

3. Presentar el diseño del software del dispositivo de monitoreo de nivel del agua potable

4. Documentar el diseño e implementación del dispositivo de monitoreo de nivel del agua potable.

5. Documentar los costos de hardware y horas de trabajo para la implementación del dispositivo.

# INTRODUCCIÓN

En la actualidad existe una gran cantidad de cisternas de almacenamiento de agua potable tanto para el sector industrial como el uso doméstico, de los cuales una gran mayoría no cuenta con un sistema de monitoreo de nivel de agua debido a su alto costo y a la poca disponibilidad en el mercado, lo cual da como resultado pérdidas económicas al momento de realizar un llenado excesivo de la cisterna o no contar con un nivel óptimo para el funcionamiento y distribución del líquido cuando este es necesario.

Los dispositivos de medición de nivel de agua potable en el mercado son clasificados según el método de medición los cuales varían según su costo, sin embargo, muchos no cuentan con la capacidad de visualizar la información en tiempo real y no permiten configurar las dimensiones de la cisterna para tener un mejor control del nivel.

Esta es la razón por la cual se presenta un dispositivo de bajo costo para el monitoreo del nivel de agua potable en cisternas a través de una aplicación web en tiempo real, el cual está diseñado por un método de medición indirecta a través de un sensor ultrasónico, que se comunica mediante una red local para actualizar la información a la aplicación web permitiendo un control del nivel del agua potable al usuario y configuración de las dimensiones de la cisterna.

# 1. TIPOS DE MEDIDORES DE NIVEL DE AGUA POTABLE

Los medidores utilizados para la medición de nivel de agua potable pueden ser clasificados según su método de operar directa o indirecta.

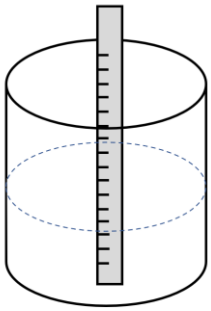
## 1.1. Medidores por método directo

Consisten en instrumentos que comparan la variable por medir con una unidad de medida de la misma naturaleza física sobre un punto de referencia.

### 1.1.1. Medidor de nivel de varilla

Este medidor está conformado por una varilla graduada en unidades de longitud convenientes, la cual es colocada dentro del contenedor del líquido. La forma de determinar el nivel es realizar una lectura directa de la superficie mojada por el líquido en contacto.

Figura 1. **Medidor de varilla**

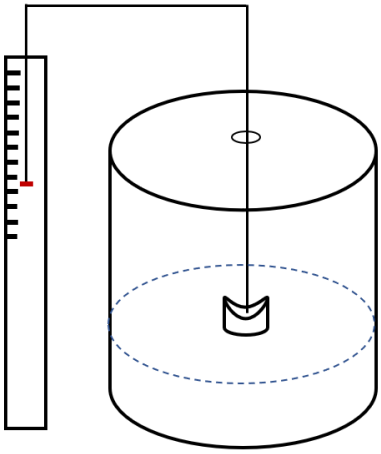


Fuente: elaboración propia, empleando eDraw 11.1.1.

### 1.1.2. Medidor de nivel de flotador

Consiste en colocar un flotador en el interior del contenedor del líquido y conectado al exterior del contenedor por medio de una cinta de longitud conveniente. El funcionamiento de este medidor es sobre una escala de longitud graduada, la cual cambia a medida que el nivel del líquido aumenta o disminuye, provocando un cambio en el flotador.

Figura 2. **Medidor de flotador**



Fuente: elaboración propia, empleando eDraw 11.1.1.

## 1.2. Medidores por método indirecto

Consisten en instrumentos que utilizan variables distintas a las de su propia naturaleza física permitiendo calcular la medición requerida, este método es utilizado debido a que se pueden presentar problemas en la magnitud del valor por medir, recursos o facilidad de realizar medición en otras variables.

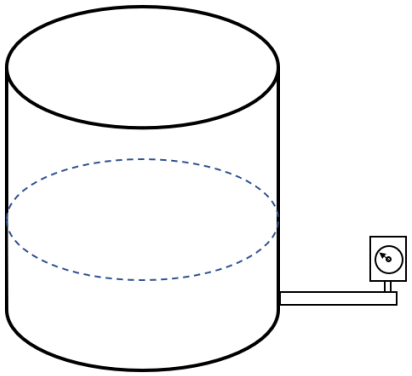
### 1.2.1. Instrumentos de presión hidrostática

Son aquellos que utilizan el valor de la presión hidrostática, la cual cambia por el efecto de la gravedad ejercido sobre el líquido al modificar su nivel.

#### 1.2.1.1. Medidor manométrico

Está basado en un manómetro conectado a la parte inferior del contenedor del líquido con la función de medir la presión según el nivel del líquido, este tipo de medidor se limita a contenedores abiertos, ya que debe ser afectado por la presión hidrostática del ambiente.

Figura 3. **Medidor manométrico**



Fuente: elaboración propia, empleando eDraw 11.1.1.

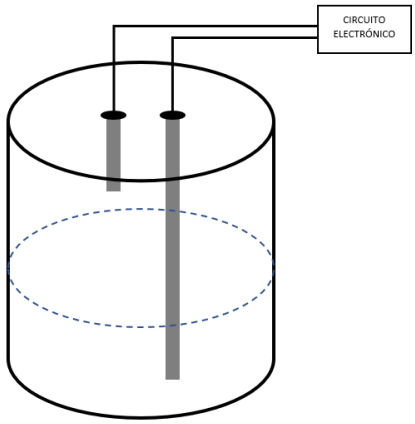
### 1.2.2. Instrumentos eléctricos

Son aquellos que utilizan valores que alteran el flujo eléctrico utilizando los cambios en la conductividad, resistencia y capacitancia.

### 1.2.2.1. Medidor conductivo resistivo

Este medidor está conformado por uno o varios electrodos los cuales al tener contacto con el líquido conductor provocan un cambio en la conductividad o resistencia, lo cual es detectado por un circuito electrónico integrado que interpreta estos cambios, enviando una señal a un actuador.

Figura 4. **Medidor conductivo resistivo**

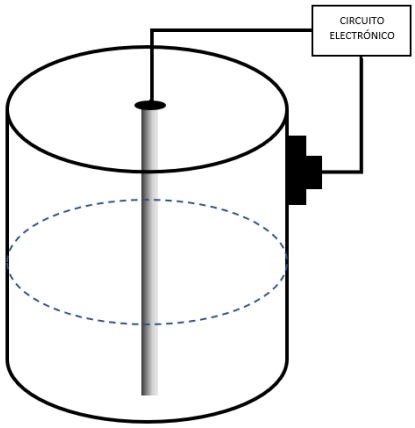



Fuente: elaboración propia, empleando eDraw 11.1.1.

### 1.2.2.2. Medidor capacitivo

Este medidor consta de un circuito electrónico conectado a un electrodo introducido al centro del contenedor y otra conexión a la pared conductora del tanque, conformando un condensador el cual varía su capacitancia según el nivel del líquido en el contenedor.

Figura 5. **Medidor capacitivo**




Fuente: elaboración propia, empleando eDraw 11.1.1.

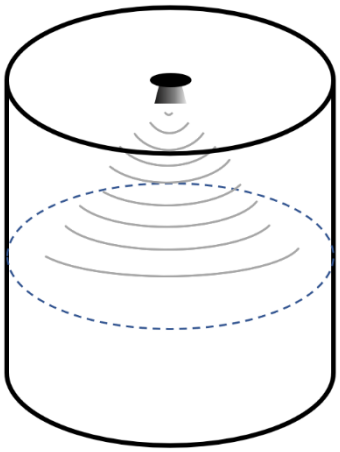
### 1.2.3. Instrumentos físicos

Son aquellos que utilizan propiedades físicas para obtener los valores de nivel por medio de una variable relacionada, en los cuales se puede considerar instrumentos por ultrasonido, radiación y láser, entre otros.

#### 1.2.3.1. Medidor por ultrasonido

Este medidor consta de un emisor y receptor, en el cual se emiten pulsos ultrasónicos los cuales son reflejados por una superficie, estos pulsos reflejados son recibidos por el receptor en un lapso, permitiendo calcular una distancia en la unidad de medida respectiva.

Figura 6. **Medidor por ultrasonido**

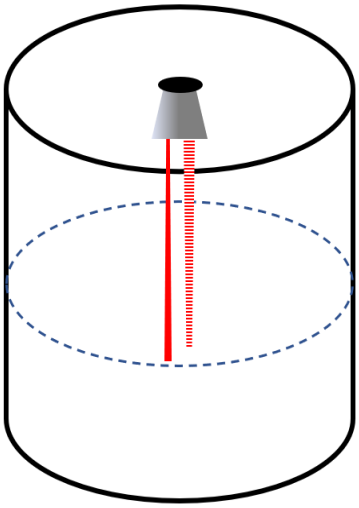


Fuente: elaboración propia, empleando eDraw 11.1.1.

#### 1.2.3.2. Medidor por láser

Este medidor consta de un emisor de rayo láser el cual es enviado en dirección del líquido generando una reflexión la cual es captada por un fotodetector, esta reflexión tiene un tiempo de duración el cual es utilizado para calcular la distancia en la unidad de medida respectiva.

Figura 7. **Medidor por láser**



Fuente: elaboración propia, empleando eDraw 11.1.1.

# 2. DISEÑO DEL HARDWARE DEL DISPOSITIVO DE MONITOREO DEL NIVEL DE AGUA POTABLE

En el presente capítulo se explica la teoría fundamental de los componentes electrónicos utilizados en la implementación y la descripción del diseño de hardware del dispositivo de monitoreo del nivel de agua potable.

## 2.1. Generalidades del diseño

El dispositivo de monitoreo de nivel de agua potable se ha desarrollado por medio de una serie de dispositivos electrónicos los cuales al ser ensamblados y con la configuración realizada cumplen el objetivo del dispositivo planteado.

El dispositivo es diseñado con la finalidad de ser portable, de fácil movilidad y adaptable a cualquier recipiente, de agua potable. Una de las características esenciales es que este pueda ser colocado en la parte superior del recipiente de agua, para tener una medición certera del nivel de agua.

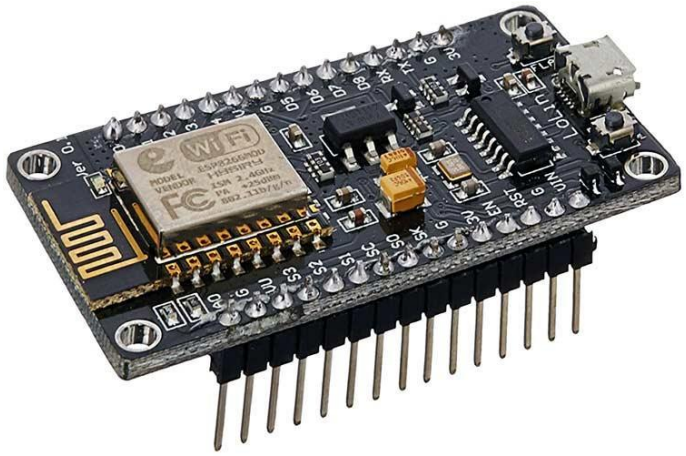
## 2.2. Descripción de los componentes electrónicos utilizados

El hardware del dispositivo consta de dos componentes fundamentales para el funcionamiento, uno es la tarjeta de desarrollo NodeMCU y el otro, el sensor ultrasónico JSN-SR04T

### 2.2.1. NodeMCU

El NodeMCU es una placa de desarrollo de código abierto que integra un microcontrolador y una tarjeta Wi-Fi ESP8266 capaz de funcionar como un servidor web cliente o servidor. Esta placa de bajo costo está compuesta por pines de entrada y salida los cuales pueden ser manipulados para controlar dispositivos periféricos digitales o analógicos según la configuración deseada. La configuración a nivel lógico de esta placa se realiza a través de un firmware adaptable a lenguajes como micro Python, Lua y Arduino.

Figura 8. **NodeMCU ESP8266**



Fuente: COMPONENTS 101. *NodeMCU ESP8266.* https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet. Consulta: 10 de julio de 2020.

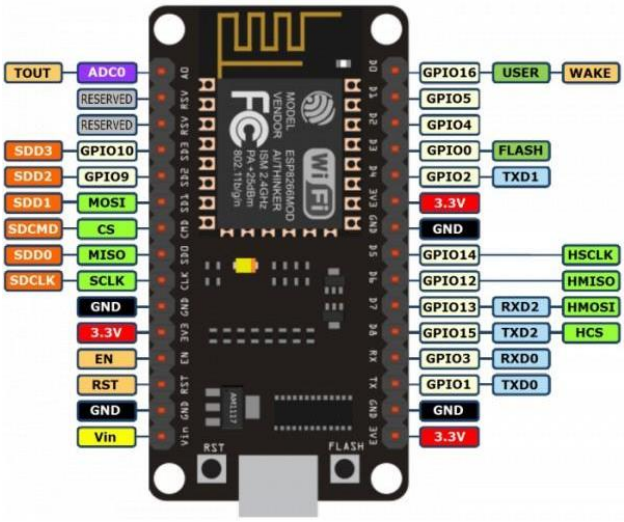
Para el funcionamiento de la tarjeta NodeMCU es necesario tener una alimentación de voltaje entre los 3,3 v hasta los 5 v, el voltaje de operación debe comprender en el rango descrito para que la eficiencia y desempeño de la placa sea óptimo.

Esta tarjeta de desarrollo cuenta con un total de 30 pines los cuales se distribuyen según su utilidad.

#### 2.2.1.1. Diagrama de distribución de pines

En el siguiente diagrama se presenta una descripción física de la tarjeta de desarrollo NodeMCU, en la cual se describe la distribución de pines y funcionalidad de cada uno según lo descrito en los puntos anteriores.

Figura 9. **Diagrama de puertos NodeMCU ESP8266**




Fuente: COMPONENTS 101. *NodeMCU ESP8266.* https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet. Consulta: 10 de julio de 2020.

#### 2.2.1.2. Descripción de pines

Cada uno de los pines tiene una función establecida desde el microcontrolador y son clasificados según esta.

#### 2.2.1.2.1. De alimentación

Esta tarjeta cuenta con 3 pines de salida de voltaje de 3,3 v, los cuales tienen como finalidad la alimentación de dispositivos externos. Adicional, cuenta con un pin de voltaje de entrada o salida llamado VIN el cual puede funcionar para alimentar dispositivos externos o alimentar la tarjeta de desarrollo con 5 v regulados.

#### 2.2.1.2.2. De tierra

Los pines de tierra son pines en común dentro de la placa de desarrollo, estos pines se conectan al valor de referencia de 0 voltios.

#### 2.2.1.2.3. De protocolo I2C

Son utilizados para conectar periféricos y sensores I2C, puede ser utilizado como I2C maestro o esclavo. El protocolo I2C se utiliza mediante la configuración de firmware y la frecuencia de reloj de la placa de desarrollo.

#### 2.2.1.2.4. De entrada y salida

Esta tarjeta de desarrollo cuenta con 17 pines que pueden ser configurados con pines de entrada y salida digital, estos se pueden habilitar como *pull-up* o *pull-down* internamente, la salida lógica de los pines de entrada y salida son configurados mediante el firmware.

#### 2.2.1.2.5. UART

Los pines UART son clasificados como TX de transmisión y RX de recepción, la tarjeta de desarrollo nodemcu cuenta con 2 interfaces UART, las cuales pueden utilizarse para comunicación con dispositivos externos que manejen el mismo protocolo de comunicación. Así mismo, el puerto micro-usb de dicha tarjeta puede utilizarse para este fin.

#### 2.2.1.2.6. SPI

Estos pines pueden ser utilizados en modo maestro y esclavo, los cuales pueden configurarse en cuatro modos de temporización de transferencia en formato SPI hasta con 80 megaciclos.

#### 2.2.1.2.7. SDIO

Estos pines son digitales de entrada y salida con seguridad, los cuales son utilizados para conectar periféricos de almacenamiento como tarjetas SD.

#### 2.2.1.2.8. PWM

La salida de modulación por ancho de pulso (PWM) es utilizada mediante programación y se emplea para la generar un nivel de voltaje análogo mediante la variación de frecuencia, la cual es ajustable entre 100Hz hasta 1kHz, la placa de desarrollo cuenta con 4 pines de modulación PWG

#### 2.2.1.2.9. De RST

Se utilizan para ejecutar las funciones preestablecidas de control de la tarjeta de desarrollo, estos pines cumplen la función de reiniciar el funcionamiento de la placa.

### 2.2.2. Sensor ultrasónico JSN-SR04T

La función de dicho sensor comprende la emisión de ondas ultrasónicas que son utilizadas para obtener la distancia entre un objeto con respecto al punto de referencia que presenta el sensor.

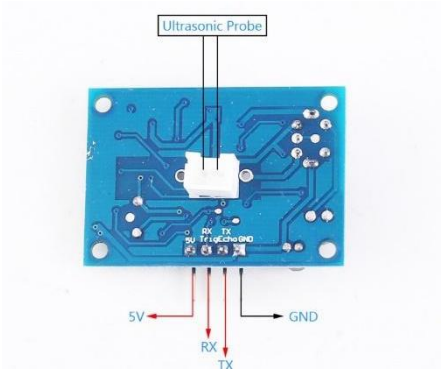
Este sensor obtiene la distancia mediante la emisión de pulsos ultrasónicos los cuales se denominan *trigger*, estos rebotan en el objeto próximo en línea recta, retornando la señal ultrasónica denominada echo a nuestro receptor en un intervalo de tiempo. La diferencia de tiempo entre el *trigger* y echo es la medición de distancia obtenida.

Una de las características esenciales para el dispositivo que presenta este sensor es la resistencia al agua y líquidos.

#### 2.2.2.1. Descripción de pines

Este sensor cuenta con cuatro pines de los cuales uno es de alimentación a 5v, uno de conexión a tierra, y los restantes de *trigger* y echo, como se observan en la siguiente figura.

Figura 10. **JSN-SR04T**

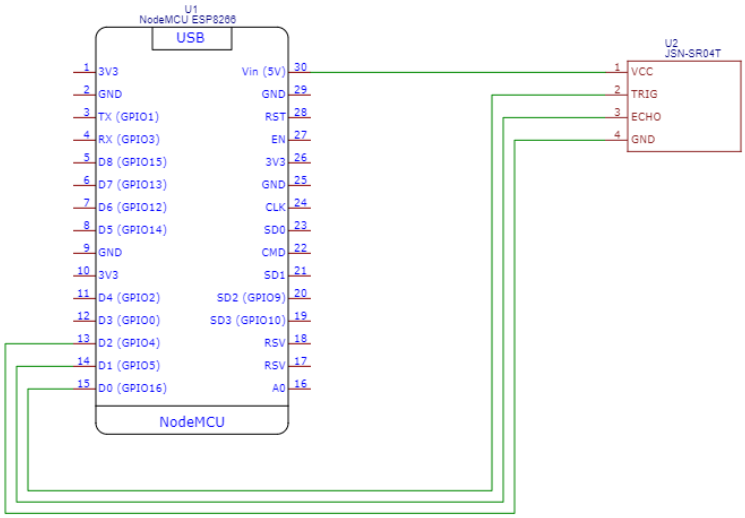



Fuente: Best Sellers. *JSN-SR04T Integrated Ultrasonic Distance Measuring Sensor Module Waterproof Transducer Sensor.* http://www.icstation.com/sr04t-integrated-ultrasonic-distance-measuring-sensor-module-waterproof-transducer-sensor-p-14739.html. Consulta: julio de 2020.

## 2.3. Interconexión de dispositivos

Para obtener la lectura del sensor JSN-SR04T e interactuar mediante la placa de desarrollo NodeMCU es necesario realizar una interconexión estableciendo comunicación entre ambos elementos. Por lo cual se debe conectar el TX y RX a un pin digital respectivamente. Para los fines establecidos se realiza la conexión del *trigger* al pin digital D1 y echo al pin digital D2, la alimentación del sensor se realizará a través del pin (Vin) que suministra 5 voltios.

Figura 11. **Diagrama de conexión**




Fuente: elaboración propia.

# 3. DISEÑO DEL SOFTWARE DEL DISPOSITIVO DE MONITOREO DEL NIVEL DE AGUA POTABLE

El diseño del software para el dispositivo de monitoreo del nivel de agua potable se conforma por dos partes esenciales, la primera es el programa de la tarjeta de desarrollo NodeMCU para la toma de distancia por medio del sensor e interpretación de los datos y como segunda parte, el programa encargado de recibir la información de la tarjeta y mostrarla de manera visual.

Para el diseño de software de la tarjeta NodeMCU se opta por el lenguaje de desarrollo Arduino el cual tiene las ventajas de ser flexible, de código abierto que permite integrar librerías para interactuar con los distintos puertos del microcontrolador y pines integrados de la tarjeta de desarrollo.

El diseño de software para obtener la información de la tarjeta NodeMCU se basa en lenguaje Python utilizando el micro framework Flask, el cual permite crear un servidor web para recibir peticiones html enviadas por la tarjeta de desarrollo. Así mismo, permite desplegar el panel de control en una página html de manera visual.

## 3.1. Diagrama de bloques del software

El diseño de software para el dispositivo se conforma por una serie de etapas, las cuales se presentan en la figura 12, se dividen en dos partes presentando de color verde las que son realizadas por la tarjeta de desarrollo NodeMCU y de color azul las etapas realizadas por el software desarrollado en el lenguaje Python desde el computador.

Figura 12. **Diagrama de bloques del software**


Fuente: elaboración propia, empleando Visio 2016.

### 3.1.1. Recepción de la señal del sensor

La recepción de la señal del sensor ultrasónico es obtenida mediante la tarjeta NodeMCU, previo a ser procesada.

Esta recepción se puede realizar al emitir un pulso alto en el pin TX del sensor, activando uno de los transductores por un intervalo de tiempo el cual emite una señal ultrasónica que es transmitida en línea recta. Posteriormente al realizar la emisión de la señal esta es recibida por el segundo transductor y dicho pulso es enviado por medio del pin RX del sensor hacia la tarjeta de desarrollo.

El programa para ejecutar las acciones necesarias para obtener las señales se inicia estableciendo la configuración de los pines digitales de entrada y salida como constantes, configurando el pin D6 como entrada y el pin D7 como salida.

En la configuración general del código se establece una comunicación serial, para desplegar a través del puerto serial la recepción de valores en el microcontrolador.

Figura 13. **Configuración inicial - código**

```
const int Trigger = D7;        // Configuración del pin D7 Trigger del sensor
const int Echo = D6;           // Configuración del pin D6 Trigger del sensor

void setup() {
  Serial.begin(9600);          // Se inicia comunicacion serial a 9600 baudios
  pinMode(Trigger, OUTPUT);    // Configuración de pin Trigger como salida
  pinMode(Echo, INPUT);        // Configuración de pin Echo como entrada
  digitalWrite(Trigger, LOW); // Se define estado inicial bajo al pin Trigger
}
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

Posterior a la configuración inicial del código, se realiza la configuración de la función principal del programa, en la cual se realizará el proceso para obtención de información del sensor en un ciclo continuo.

Inicialmente se define una variable t, la cual será utilizada para obtener el valor del tiempo en microsegundos transcurridos desde el envío de la señal por el pin *trigger* y la recepción por el pin echo.

Figura 14. **Función principal - código**

```
void loop()
{

  long t;                     // Tiempo transcurido en retornar la señal
  digitalWrite(Trigger, HIGH);// Envio de pulso de estado alto en el pin Trigger
  delayMicroseconds(10);      // Retardo de 10 microsegundos
  digitalWrite(Trigger, LOW); // Finaliza el estado alto en el pin Trigger
  t = pulseIn(Echo, HIGH);    // Obtenemos el timpo de retorno del pulso alto en microsegundos
  Serial.println(t);          // Se envia el valor obtenido por medio de puerto serial
  delay(100);                 // Pausa de 100ms
}
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

La información obtenida en la tarjeta NodeMCU será transmitida a la computadora por medio del bus serial para ser visible para el usuario. Al finalizar esta acción nuevamente se ejecutará la función principal en un intervalo de 10 milisegundos.

### 3.1.2. Procesamiento de la señal

La señal en forma de pulso digital recibida por la tarjeta NodeMCU, es procesada de manera que sea un valor medible y práctico para el desarrollo, por lo cual se realiza una conversión para obtener como resultado un valor numérico en metros.

El valor de tiempo en microsegundos en la etapa de recepción de la señal del sensor es una medición indirecta de la distancia del objeto que refleja la señal enviada por el Trigger. Esta medición debe ser procesada con la finalidad de obtener el valor de distancia desde el punto de origen hacia el objeto próximo en línea recta.

Para calcular el valor de la distancia se parte del concepto de la velocidad media el cual la define como directamente proporcional a la distancia recorrida e inversamente proporcional al tiempo transcurrido, de esta manera se define que la unidad de velocidad debe ser en metros/segundo.

$$velocidad = \frac{distancia}{tiempo} \qquad (3:1)$$

En donde la velocidad es la velocidad del sonido 340 m/s, para fines prácticos se realiza la conversión para centímetros sobre segundos ya que el valor obtenido por el sensor es en microsegundos. Debido a que la señal llega al

objeto destino y esta es reflejada la distancia recorrida es el doble modificando la ecuación 3:1 dando como resultado la ecuación 3.2.

$$\frac{340m}{s} * \frac{1s}{1\,000\,000\mu s} * \frac{100cm}{1m} = \frac{2 * distancia}{tiempo} \qquad (3:2)$$

Así resulta la ecuación 3.3 que indica que la distancia obtenida por el sensor ultrasónico en centímetros será directamente proporcional al tiempo en microsegundos e inversamente proporcional a la constante de velocidad 59 en centímetros sobre segundos.

$$distancia = \frac{t}{59}\ [centimetros] \qquad (3:3)$$

Agregando esta ecuación al código de la figura 15, se realiza el proceso en la función principal de manera de que el valor final sea la distancia en centímetros.

Figura 15. **Procesamiento de la señal - código**

```
void loop()
{

  long t;                    // Tiempo transcurido en retornar la señal
  long d;                    // Distancia recorrida por la señal
  digitalWrite(Trigger, HIGH);// Envio de pulso de estado alto en el pin Trigger
  delayMicroseconds(10);     // Retardo de 10 microsegundos
  digitalWrite(Trigger, LOW); // Finaliza el estado alto en el pin Trigger
  t = pulseIn(Echo, HIGH);   // Obtenemos el timpo de retorno del pulso alto en microsegundos
  d = t/59;                  // Calculo de la distancia
  Serial.println(d);         // Se envia el valor obtenido por medio de puerto serial
  delay(100);                // Pausa de 100ms
}
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

### 3.1.3. Comunicación cliente servidor

En esta etapa se realiza la configuración de la tarjeta NodeMCU como cliente de una red local, capaz de enviar información a un servidor por medio de peticiones HTTP, de esta manera se podrá enviar la información de nivel en metros obtenida para ser manipulada y desplegada visualmente en la siguiente etapa.

Para la creación de la comunicación cliente servidor es necesario conectar la tarjeta NodeMCU a una red local, esta conexión se realizará de manera inalámbrica por medio de la señal wifi de una red local estableciendo una dirección estática de manera de que esta no cambie cada vez que el dispositivo se conecta a la red.

La configuración de red para la tarjeta se realiza por medio de la importación de la librería ESP8266WiFi la cual permite realizar la configuración de parámetros del módulo de red integrado ESP8266 y la librería ESP8266HTTPClient que permite la configuración del cliente http.

Inicialmente se realiza la configuración del nombre de red, clave y la configuración de la dirección ip estática que se asignará a la tarjeta, dirección de puerta de red (gateway) y mascará de subred (subnet) como se muestra en la figura 16 con los siguientes parámetros:

- Dirección IP: 192.168.1.200
- Dirección de Red: 192.168.1.1
- Mascara de subred: 255.255.255.0

Figura 16. **Configuración cliente wifi – código**

```
#include <ESP8266HTTPClient.h>
#include <ESP8266WiFi.h>

const char *nombre_red = "WLAN";                    // Establecer nombre de la Red Wifi
const char *clave = "ABCDE7485";                    // Establecer el password de la Red Wifi

IPAddress ip(192,168,1,200);                        // Establecer direccion ip estatica
IPAddress gateway(192,168,1,1);                     // Establecer puerta de enlace
IPAddress subnet(255,255,255,0);                    // Establecer mascara de subred
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

En la configuración principal del código se establecerá la conexión de la tarjeta a la red local, con las constantes definidas en la figura 17 iniciando un ciclo que finalice hasta que la conexión sea exitosa, al finalizar esta conexión se enviará al puerto serial el resultado de la conexión establecida y se mostrará la dirección ip asignada.

Figura 17. **Configuración inicial cliente wifi – código**

```
void setup()
{

  Serial.begin(9600);                               // Configurar comunicación serial 9600 baudios
  WiFi.mode(WIFI_STA);                              // Configuracion comunicación Wifi estatica
  WiFi.config(ip, gateway, subnet);                 // Se configura la red Wifi
  WiFi.begin(ssid, password);                       // Se inicializa conexión a la red Wifi

  while (WiFi.status() != WL_CONNECTED)             // Se espera el estado de la conexión
  {
    delay(200);
    Serial.print('.');
  }

  Serial.println("Conexión establecida");           // Se envia el estado de la conexión al ser establecida
  Serial.println(WiFi.localIP());                   // Se imprime la direccion ip asignada.
}
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

La configuración inicial del código tiene como finalidad ejecutar una consulta por medio del método post a la ruta definida del servidor, para dicha funcionalidad se establece el cliente HTTPClient y WiFiClient, para iniciar las consultas a las rutas específicas, que pueden traer valores de retorno los cuales pueden ser procesados y utilizados para realizar acciones en función de estos.

Para este caso se enviarán dos consultas con valores estáticos configurados en el código, siendo para la primera consulta el dato 25 a la ruta valor y en la segunda consulta el valor de tiempo a la ruta tiempo el cual es almacenado en la variable tiempo de tipo entero, con la finalidad de utilizar el valor para definir el intervalo en segundos entre cada ciclo. El resultado de esta configuración se observa en la figura 18 con los siguientes parámetros:

- Ruta para enviar dato: http://192.168.1.8:8090/valor
- Ruta para obtener el tiempo: http://192.168.1.8:8090/tiempo

Figura 18. **Función principal cliente wifi – código**

```
void loop()
{


   HTTPClient http;                                                    // Se establece cliente http
   WiFiClient client;                                                  // Se establece cliente Wifi

    http.begin(client,"http://192.168.1.8:8090/valor");                // Se configura la direccion del Servidor y la ruta
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");// Se configura el tipo de contenido a enviar
    http.POST("dato="+String(25));                                     // Se envia el valor de deistancia
    http.end();                                                        // Se finaliza cliente http
    http.begin(client,"http://192.168.1.8:8090/tiempo");               // Se configura la direccion del Servidor y la ruta
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");// Se configura el tipo de contenido a enviar
    int tiempo = http.POST("tiempo=tiempo");                           // Se realiza la consulta el valor de tiempo
    http.end();                                                        // Se finaliza cliente http
    Serial.println(tiempo*60*1000/100);                                // Se convierte el tiempo a minutos
    delay(tiempo*60*1000/100);                                         // Retardo en minutos
}
```

Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

Posterior a la configuración del cliente es necesario agregar la configuración de obtención de señal del sensor y procesamiento de señal, dando como

resultado el diseño final de software configurable en la tarjeta de desarrollo NodeMCU. Este código se puede observar en el área de apéndices.

### 3.1.4. Recepción de la información

En esta etapa inicia la configuración de lado del servidor y aplicación web, de manera que la tarjeta de desarrollo pueda comunicarse al servidor y enviar las peticiones http con los valores correspondientes. Para esta configuración se implementa el desarrollo en lenguaje Python para lo cual se crea el archivo tipo Python para iniciar el desarrollo del código.

Inicialmente se debe importar el micro-framework Flask el cual permite realizar la configuración del servidor, rutas, renderizar plantillas, enviar peticiones http y recepción de valores. Adicional se realiza la configuración inicial de la aplicación en la cual se define el *host* de salida y el puerto de la aplicación como se observa en la figura 19.

Figura 19. **Configuración inicial servidor python – código**

```
1  from flask import Flask, request,render_template
2  app = Flask(__name__)
3  app.run(host='0.0.0.0', port=8090,debug=True)
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

#### 3.1.4.1. Creación de la base de datos

Al realizar la configuración inicial del servidor es necesario iniciar con el desarrollo del proyecto, empezando con la configuración de base de datos sqlite3 para almacenar la información de los parámetros del tanque y valores de nivel en función del tiempo. Creando una base de datos con nombre base_de_datos.db por medio de sqlite3. La base de datos tendrá dos tablas nombradas nivel_de_agua y valores.

En la tabla nombrada nivel_de_agua se almacenará el valor en centímetros del nivel de agua y la fecha en la cual se realizó la captura de la medición. La estructura de la tabla se realizará con los siguientes parámetros:

- Columna “fecha” de tipo TEXT
- Columna “valor” de tipo REAL

Para la creación de la tabla se realizará la utilización del siguiente comando sql:

```
CREATE TABLE IF NOT EXISTS nivel_de_agua( fecha TEXT, valor REAL)
```

La tabla nombrada valores en la cual se almacenará el tiempo entre toma de datos y las dimensiones del tanque altura, ancho y largo en la cual la estructura de la tabla será la siguiente:

- Columna “tiempo” de tipo REAL
- Columna “altura” de tipo REAL
- Columna “ancho” de tipo REAL
- Columna “largo” de tipo REAL

Para la creación de la tabla se realizará la utilización del siguiente comando sql:

```
CREATE TABLE IF NOT EXISTS
valores( tiempo REAL,altura REAL,ancho REAL,largo REAL)
```

Para crear la base de datos en sqlite3 a través del código desarrollado en Python y la creación de bases de datos se realiza por medio de un conector el cual permite la modificación e interactuar con el archivo que almacena la base de datos. Dicho código se observa en la figura 20.

Figura 20. **Configuración base de datos – código**

```python
from flask import Flask, request,render_template
import sqlite3

app = Flask(__name__)

conector = sqlite3.connect('base_de_datos.db')
cursor = conector.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS nivel_de_agua( fecha TEXT, valor REAL)")
cursor.execute(
"CREATE TABLE IF NOT EXISTS valores( tiempo REAL,altura REAL,ancho REAL,largo REAL)")
conector.close()


app.run(host='0.0.0.0', port=8090,debug=True)
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

#### 3.1.4.2. Creación de la ruta para obtener distancia

La configuración de rutas es fundamental para un servidor, ya que es el medio por el cual este presenta la información a través de las peticiones http. Para recibir la información de la distancia desde la tarjeta NodeMCU, se creará

la ruta valor la cual obtendrá una variable de nombre dato por medio de una petición POST.

La ruta valor llamará una función de nombre obtener_valor, la cual verifica que se esté recibiendo un valor por medio del método POST, se almacena el valor obtenido en la variable dato y este es procesado para ser almacenado en la base de datos con la fecha en la cual este fue obtenido. La configuración de esta función se observa en la figura 21.

Figura 21. **Configuración de ruta valor – código**

```
@app.route('/valor', methods=['POST', 'GET'])
def obtener_valor():
    if request.method == 'POST':
        dato=request.form['dato']
        ahora = datetime.now()
        hora_fecha = ahora.strftime("%Y-%m-%d %H:%M:%S")
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("INSERT INTO nivel_de_agua VALUES('"+hora_fecha+"','"+dato+"')")
        conector.commit()
        cursor.close()
        conector.close()
        return "Recibido"
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

#### 3.1.4.3. Creación de la ruta para obtener tiempo

La tarjeta NodeMCU tiene como función obtener el tiempo desde la ruta tiempo, dicho valor es una variable que define el lapso en minutos. Para la toma de datos de nivel este valor es configurable y es obtenido desde la base de datos y retornado desde esta función.

La función tiene como objetivo realizar una consulta a la base de datos valores y obtener el valor de tiempo, para ser retornado por medio de una petición POST desde la ruta tiempo, dicha configuración se observa en la figura 22.

Figura 22. **Configuración de ruta tiempo – código**

```python
@app.route('/tiempo', methods=['POST', 'GET'])
def tiempo():
    if request.method == 'POST':
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("SELECT * FROM valores")
        valores = cursor.fetchone()
        cursor.close()
        conector.close()
        tiempo=valores[0]
        return str(tiempo)
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

### 3.1.5. Etapa de visualización

El diseño de la etapa de visualización es una parte fundamental para la interactividad con el usuario final, por lo cual debe ser una interfaz simple de fácil uso y amigable con el usuario.

Esta etapa se encuentra en la parte del desarrollo que se realiza en el servidor por medio del desarrollo en Python, por lo cual se hace uso nuevamente del micro-framework Flask el cual permite realizar mediante sus funciones la definición de rutas e integración de las vistas en lenguaje HTML.

#### 3.1.5.1. Creación de la ruta principal

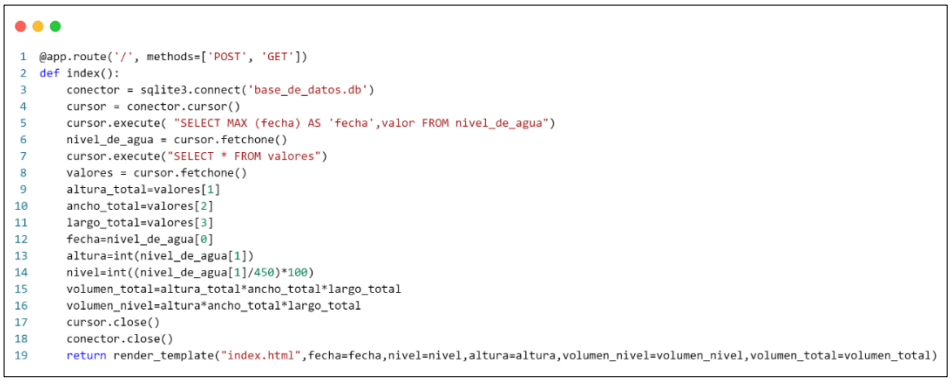
La ruta principal es la dirección del servidor, al ingresar a esta dirección se mostrará la visualización del tanque y la representación gráfica del nivel que posee correspondiente al último dato registrado en la base de datos.

Para el diseño de la ruta principal es necesario realizar una serie de pasos para hacer el despliegue de la información los cuales son:

- Consulta de la fecha y último valor de profundidad registrado en la tabla niveles_de_agua.
- Consulta de las dimensiones del tanque registrados en la tabla valores.
- Se calcula el porcentaje de llenado en función de la altura total obtenida de la base de datos.
- Se calcula el volumen del tanque en función de los datos obtenidos en la base de datos.
- Se calcula el volumen del nivel actual en función del valor de profundidad registrado en la base de datos y las dimensiones del tanque.
- Se renderiza la plantilla html que mostrará gráficamente la información del tanque.
- Se envían los valores de fecha, nivel, volumen del nivel y volumen total al momento de renderizar la plantilla html.

Al tomar las funciones descritas la creación de la función principal se observa en la figura 23.

Figura 23. **Configuración de ruta principal – código**



```
@app.route('/', methods=['POST', 'GET'])
def index():
    conector = sqlite3.connect('base_de_datos.db')
    cursor = conector.cursor()
    cursor.execute( "SELECT MAX (fecha) AS 'fecha',valor FROM nivel_de_agua")
    nivel_de_agua = cursor.fetchone()
    cursor.execute("SELECT * FROM valores")
    valores = cursor.fetchone()
    altura_total=valores[1]
    ancho_total=valores[2]
    largo_total=valores[3]
    fecha=nivel_de_agua[0]
    altura=int(nivel_de_agua[1])
    nivel=int((nivel_de_agua[1]/450)*100)
    volumen_total=altura_total*ancho_total*largo_total
    volumen_nivel=altura*ancho_total*largo_total
    cursor.close()
    conector.close()
    return render_template("index.html",fecha=fecha,nivel=nivel,altura=altura,volumen_nivel=volumen_nivel,volumen_total=volumen_total)
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

#### 3.1.5.2. Creación de la plantilla principal

La plantilla principal es una parte fundamental de la etapa de visualización ya que es la que el usuario podrá visualizar. Para el desarrollo se inicia con la configuración inicial de la estructura de una página web html definida en la figura 24 donde se observa la estructura del archivo. El nombre de la aplicación web se define como “Visualización de nivel”, y el código para el despliegue de la aplicación será desarrollado en la etiqueta “body”.

Figura 24. **Configuración de plantilla html – código**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Visualización de nivel</title>
</head>
<body>

</body>
</html>
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

En el cuerpo del código se define una serie de clases que mostrarán la representación gráfica del tanque, los valores de volumen total, volumen actual, altura, porcentaje y fecha de la última actualización. Estos datos son obtenidos mediante registros utilizando el motor de plantillas jinja2 el cual es implementado por el micro-framework flask. En el código principal se agregarán dos botones con la ruta formulario y la ruta principal, con la finalidad de redireccionar y actualizar la página.

La estructura del código del cuerpo de la plantilla html se muestra en la figura 25.

Figura 25. **Configuración de aplicación de nivel – código**

```
<body>
    <div class="content2">
        <div class="content">
            <div class="cuadrado">
                <div class="agua" id="agua" style="height: {{ nivel }}%;">
                    <div class="fecha">Volumen Total: {{ volumen_total }} cm^2</div>
                    <div class="fecha">Volumen Actual: {{ volumen_nivel }} cm^2</div>
                    <div class="fecha">Altura: {{ altura }} cm</div>
                    <div class="fecha">Porcentaje: {{ nivel }}%</div>
                    <div class="fecha">Fecha: {{ fecha }}</div>
                </div>
            </div>
            <div>
                <a href="/" class="button">Actualizar</a>
                <a href="/formulario" class="button">Configurar Valores</a>
            </div>
        </div>
    </div>
```

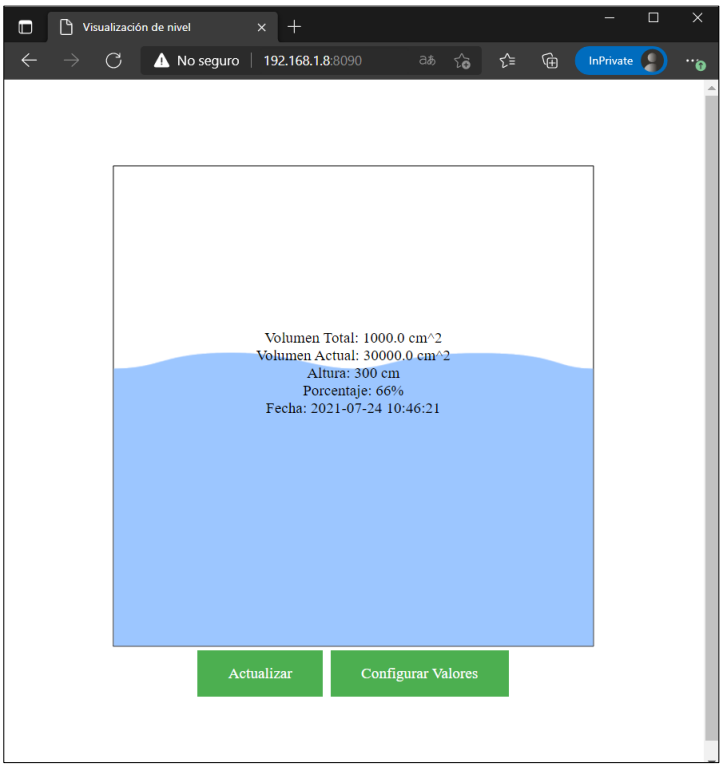
Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

El código de la plantilla html debe ser estilizado mediante código css de tal manera de que sea visualmente llamativo para el usuario. En este código se definen los siguientes estilos:

- Estilo del cuerpo de la plantilla
- Estilo y parámetros de los contenedores
- Estilo y diseño del tanque y nivel de agua
- Posición para la clase fecha utilizada para despliegue de datos
- Estilo de los botones utilizados para ejecutar rutas y formulario
- Estilo de los cuadros de entrada de texto del formulario

Aplicando la plantilla HTML con la hoja de estilos css se obtiene el diseño final de la aplicación de nivel el cual se observa en la figura 26.

Figura 26. **Diseño final de la plantilla principal – código**




Fuente: elaboración propia, empleando Chrome Edge.

El código final de estilos css y html se puede observar en el apartado de apéndices.

### 3.1.5.3. Creación de la ruta formulario

La ruta formularia tiene como función principal recibir una serie de parámetros por el usuario, los cuales permiten definir las dimensiones del tanque y el intervalo de tiempo entre mediciones de nivel. Los datos que serán capturados en la ruta serán los siguientes:

- Tiempo en minutos
- Altura en centímetros
- Ancho en centímetros
- Largo en centímetros

Estos valores serán obtenidos en la ruta formulario por medio de peticiones POST, los cuales serán procesados y almacenados en la tabla valores de la base de datos sqlite3.

Los datos del tanque deben ser valores únicos en la base de datos, por lo cual es necesario eliminar los datos de la tabla cada vez que se realice el registro de nueva información, de esta manera se tendrá seguridad de que los valores consultados sean los ingresados por el usuario.

La ruta “formulario” llamará a la función formulario la cual se define en la figura 27 y muestra el proceso descrito.

Figura 27. **Configuración de ruta formulario – código**

```python
@app.route("/formulario",methods=["POST","GET"])
def formulario():
    if request.method == "POST":
        tiempo = request.form.get("tiempo")
        ancho = request.form.get("ancho")
        largo = request.form.get("largo")
        altura = request.form.get("altura")
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("DELETE FROM valores")
        conector.commit()
        cursor.execute("INSERT INTO valores VALUES("+tiempo+","+altura+","+ancho+","+largo+")")
        conector.commit()
        cursor.close()
        conector.close()
    return render_template('formulario.html')
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

La ruta “formulario” retorna la renderización de la plantilla formulario la cual dará visualización al usuario para el ingreso de la información requerida.

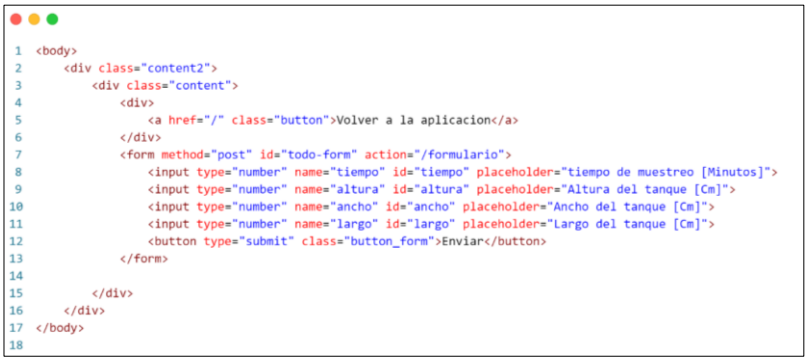
Con la creación de la ruta formulario finaliza la configuración en la aplicación desarrollada en Python, el código completo se puede observar en el apartado de apéndices.

#### 3.1.5.4. Creación de la plantilla formulario

Para la creación de la plantilla formulario se utiliza la estructura de código HTML mostrada en la figura 24, en esta estructura se realiza la definición de etiquetas en el apartado de “body”.

La plantilla “formulario” debe tener las características de mostrar campos de entrada al usuario para ingresar la información requerida y enviar los datos por medio del método post a la ruta formulario. La plantilla debe contar con cuatro datos de entrada de tipo número los cuales se definen con el nombre de tiempo, altura, ancho y largo.

Figura 28. **Configuración de plantilla formulario – código**



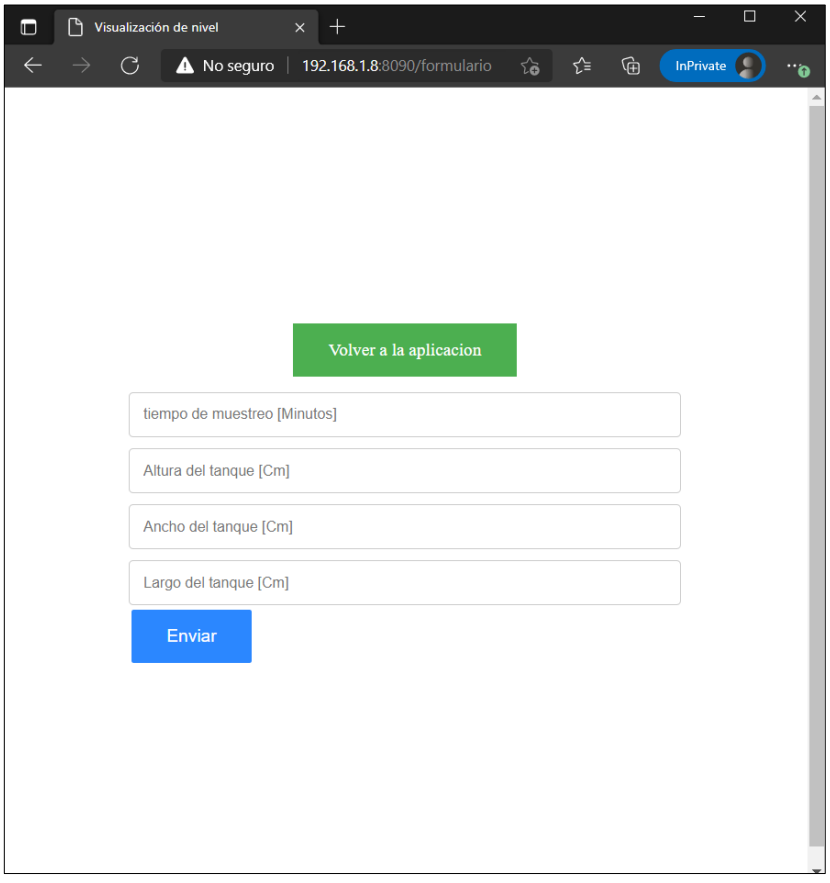
```
<body>
    <div class="content2">
        <div class="content">
            <div>
                <a href="/" class="button">Volver a la aplicacion</a>
            </div>
            <form method="post" id="todo-form" action="/formulario">
                <input type="number" name="tiempo" id="tiempo" placeholder="tiempo de muestreo [Minutos]">
                <input type="number" name="altura" id="altura" placeholder="Altura del tanque [Cm]">
                <input type="number" name="ancho" id="ancho" placeholder="Ancho del tanque [Cm]">
                <input type="number" name="largo" id="largo" placeholder="Largo del tanque [Cm]">
                <button type="submit" class="button_form">Enviar</button>
            </form>

        </div>
    </div>
</body>

```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

Adicional a la definición de clases de entrada se realiza la definición de un botón con la ruta principal para poder regresar a la aplicación principal y un botón para enviar los datos ingresados al formulario. Esta plantilla utiliza la hoja de estilos de la sección creación de plantilla principal. El código de la plantilla formulario se puede observar en el apartado de apéndices.

El diseño final de la plantilla por formular se observa en la figura 29.

Figura 29. **Diseño final de la plantilla formulario – código**

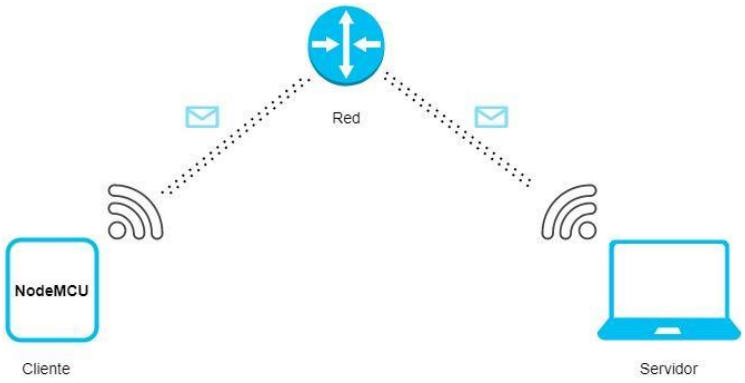



Fuente: elaboración propia, empleando Chrome Edge.

## 3.2. Diagrama de comunicación cliente servidor

La comunicación entre el cliente configurado en la tarjeta de desarrollo NodeMCU y el servidor configurado en lenguaje Python en la computadora que cumple función de servidor, se representa de manera gráfica en la figura 30 la red creada por estos dispositivos.

Figura 30. **Diagrama de comunicación cliente servidor**




Fuente: elaboración propia, empleando Chrome Edge.

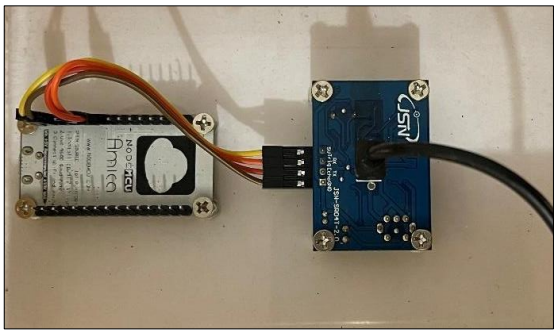
# 4. PROPUESTA DE DISEÑO E IMPLEMENTACIÓN DEL DISPOSITIVO DE MONITOREO DEL NIVEL DE AGUA POTABLE

A continuación, se describe el funcionamiento del dispositivo de monitoreo del nivel de agua potable y se documenta el diseño final propuesto e implementación.

## 4.1. Diseño del montaje del dispositivo

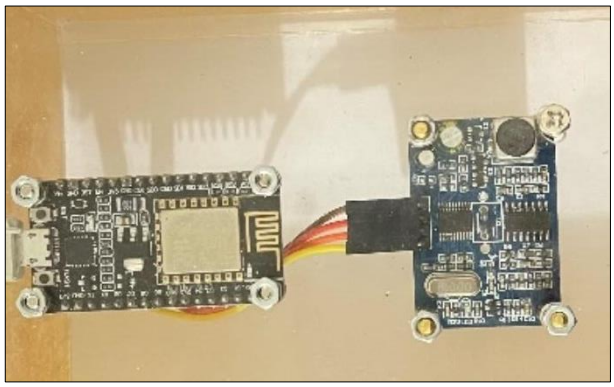
La figura 31 muestra la vista superior del dispositivo de monitoreo del nivel de agua potable y la figura 32 muestra la vista inferior del dispositivo. Los componentes son ubicados de manera que el ensamblaje sea compacto. El microcontrolador será alimentado por el conector micro-usb integrado a la placa, y el sensor conectado en la parte superior del dispositivo, el cual facilitará el mantenimiento del dispositivo.

Figura 31. **Montaje de dispositivo, vista superior**



Fuente: elaboración propia.

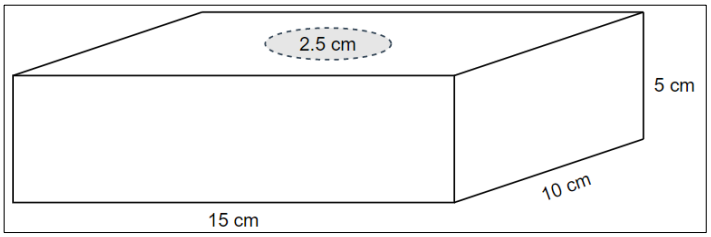
Figura 32. **Montaje de dispositivo, vista inferior**



Fuente: elaboración propia.

Todos los componentes se colocan dentro de una caja transparente de acrílico la cual es diseñada con las dimensiones y características necesarias para el aprovechamiento de espacio. En la figura 33 y 34 se muestra el diseño de la caja del dispositivo.

Figura 33. **Diseño del montaje**




Fuente: elaboración propia.

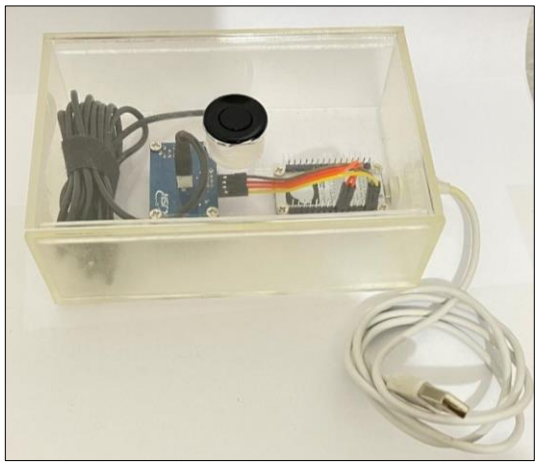
Figura 34. **Implementación del montaje**

Fuente: elaboración propia.

## 4.2. Montaje final del dispositivo

El montaje final del dispositivo corresponde a la colocación de los componentes, y el módulo ultrasónico colocado en la parte superior y con la conexión del cable de alimentación en la parte lateral de la caja. El montaje final del dispositivo se observa en la figura 35.

Figura 35. **Montaje final del dispositivo**



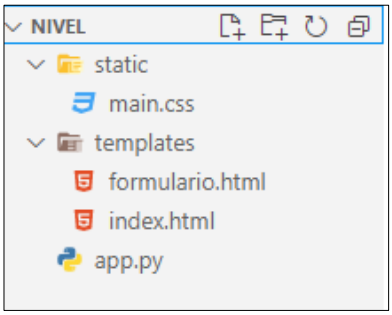
Fuente: elaboración propia.

## 4.3. Implementación del servidor

Para la implementación del servidor es fundamental la utilización de los archivos de plantillas, hojas de estilo y código Python realizados en el diseño de software del dispositivo.

Es necesario definir la estructura de archivos y carpetas mostrada en la figura 36 la cual es fundamental para el correcto funcionamiento del servidor.

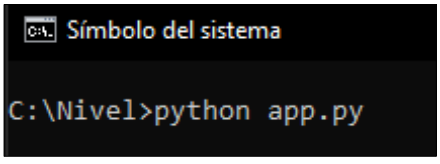
Figura 36. **Estructura de archivos**




Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

Posterior a definir la estructura de los archivos, se debe compilar el código del servidor con el siguiente comando.

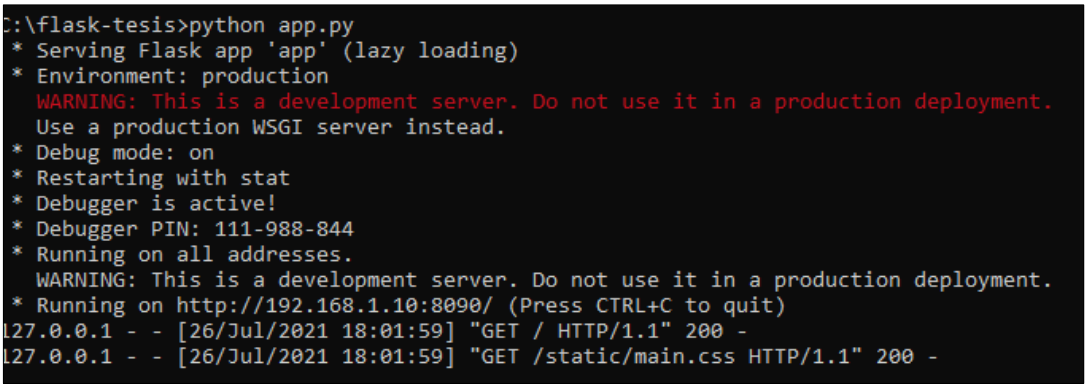
Figura 37. **Complicación código Python**




Fuente: elaboración propia, empleando símbolo del sistema de Windows 10.

Al compilar el código del proyecto, este automáticamente realiza la ejecución del servidor en el puerto 8090. Por medio de la ejecución en consola se puede observar que el proyecto está en correcto funcionamiento.

Figura 38. **Ejecución exitosa del servidor**



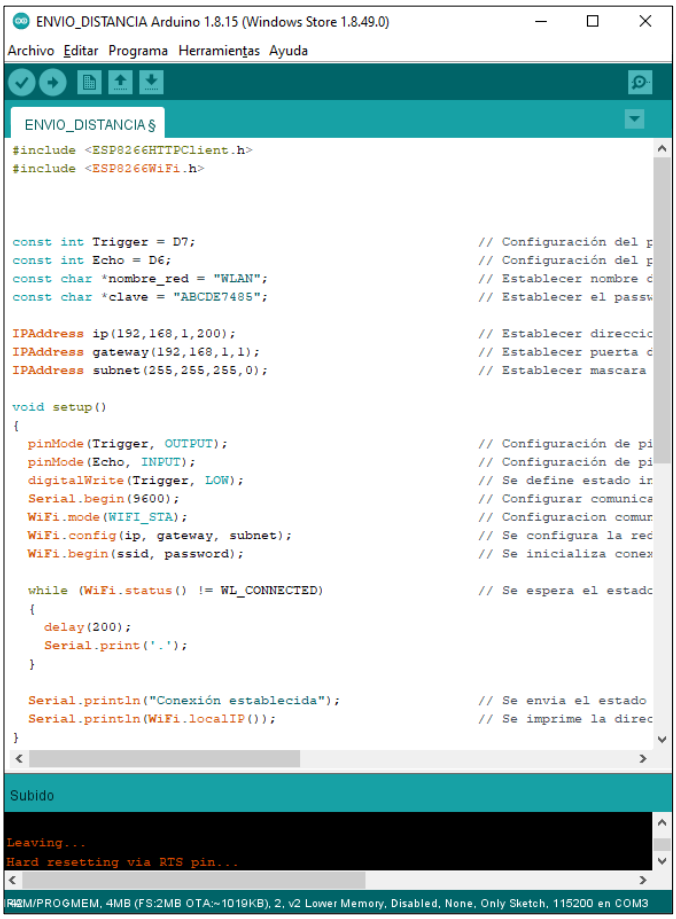

Fuente: elaboración propia, empleando símbolo del sistema de Windows 10.

## 4.4. Implementación del software del dispositivo

La implementación del software del dispositivo es la parte funcional del hardware por lo cual es esencial la configuración del software para el correcto funcionamiento. Este software es cargado de forma de código a la tarjeta de desarrollo NodeMCU por medio del programa Arduino. El software se ejecutará cada vez que la tarjeta NodeMCU sea encendida o alimentada por la fuente.

En la figura 39 se muestra la vista principal del programa Arduino, para configurar el programa en la tarjeta únicamente se debe conectar la tarjeta a la computadora y presionar el botón subir del programa Arduino.

Figura 39. **Implementación de código en Arduino IDE**




Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

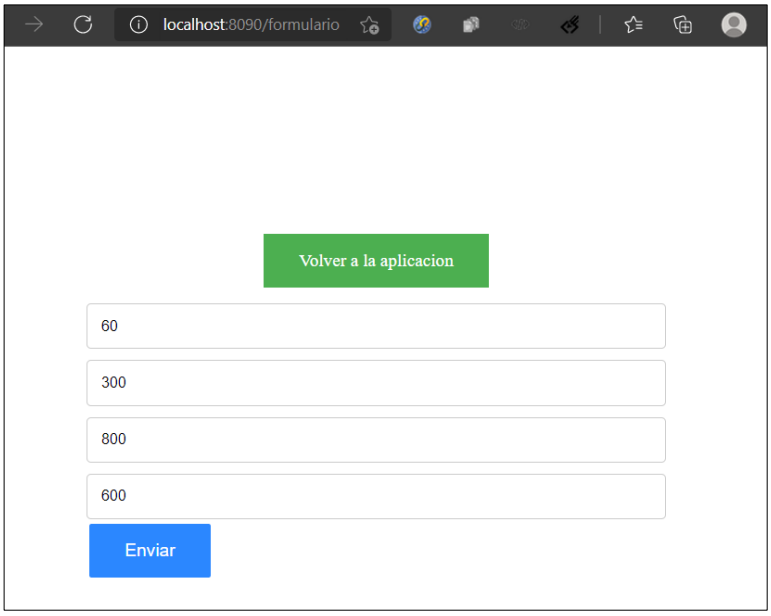
## 4.5. Pruebas de funcionamiento del dispositivo

Al finalizar la implementación del dispositivo se realizaron una serie de pruebas en cada una de las interfaces de este, por lo que se realiza una presentación de los resultados satisfactorios de funcionamiento.

### 4.5.1. Pruebas de configuración de valores

Para iniciar con las pruebas del dispositivo, es necesario configurar las dimensiones de la cisterna de agua potable en el cual se realizarán las mediciones. Estas mediciones serán ingresadas por medio del formulario definido en la ruta del mismo nombre por medio de la dirección del servidor. Para este caso se ingresará un intervalo de tiempo entre mediciones de 60 minutos, altura de 300 centímetros, ancho de 800 centímetros y largo de 600 centímetros.

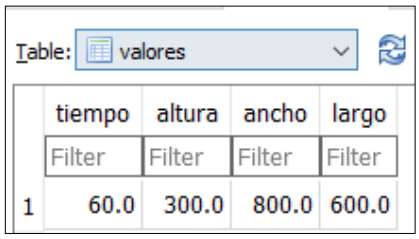
Figura 40. **Prueba de funcionamiento plantilla formulario**




Fuente: elaboración propia, empleando Microsoft Edge.

La verificación de estos datos se realizará por medio de una inspección de la base de datos en donde se confirma que estos sean procesados y almacenados en esta desde la aplicación en Python.

Figura 41. **Prueba de almacenamiento de datos en tabla valores**

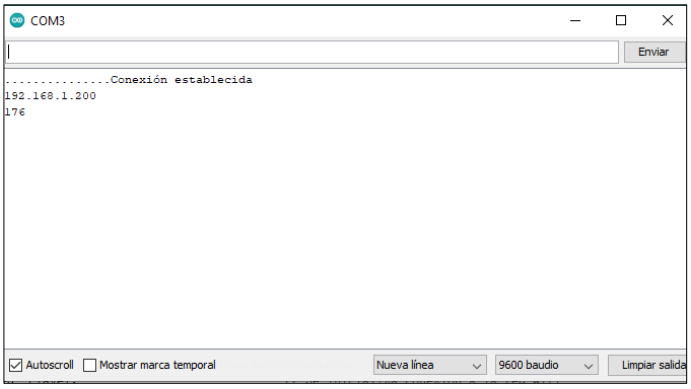



Fuente: elaboración propia, empleando SQLite 3.36.

### 4.5.2. Pruebas de funcionamiento en la tarjeta de desarrollo

El funcionamiento de la tarjeta NodeMCU se puede verificar por medio del puerto serial a través de la conexión por cable micro-usb a la computadora. Empleando el monitor serial desde el programa Arduino se verán reflejados los datos de la conexión establecida y los datos de temperatura en el rango de tiempo designado en la base de datos.

Figura 42. **Prueba de funcionamiento en Arduino IDE**

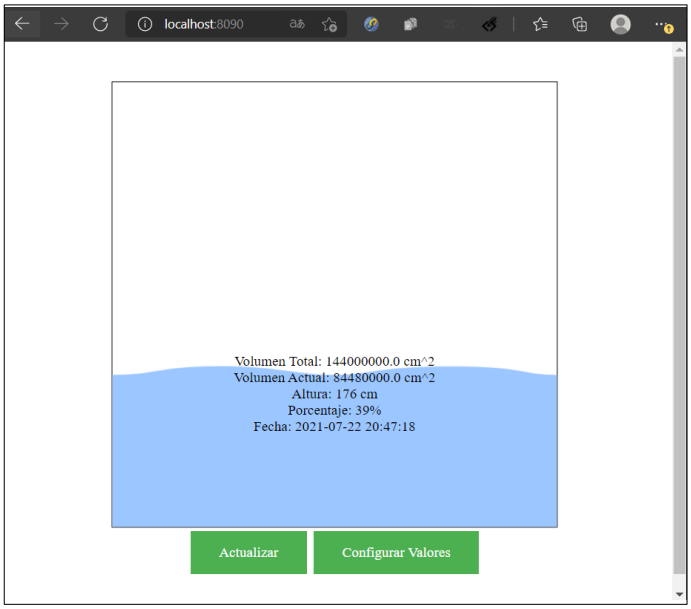



Fuente: elaboración propia, empleando Arduino IDE 1.8.16.

### 4.5.3. Pruebas de funcionamiento de la interfaz gráfica

La interfaz gráfica es la parte más importante del dispositivo, ya que en esta se muestra toda la información obtenida por el dispositivo y será la parte en la cual el usuario interactúa directamente con el dispositivo. El funcionamiento de dicha interfaz se verifica por medio de la aplicación web a la cual se puede acceder por medio de la dirección del servidor, a través de la ruta principal de este y el puerto 8090.

Figura 43. **Prueba de funcionamiento en la ruta principal**



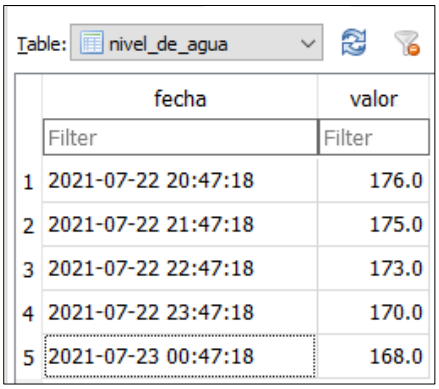

Fuente: elaboración propia, empleando Microsoft Edge.

En la figura 43 se observa el dato de nivel de líquido registrado en la figura 42 así como los datos de volumen total, volumen actual y porcentaje en función de los datos insertados en el formulario.

### 4.5.4. Pruebas de funcionamiento de almacenamiento de registros

En el formulario de parámetros se realizó la definición de intervalo de tiempo de toma de mediciones de 60 minutos lo cual se ve registrado en la figura 44 con registros en el intervalo indicado.

Figura 44. **Pruebas de almacenamiento en la tabla nivel de agua**




Fuente: elaboración propia, empleando SQLite 3.36.

# 5. COSTOS

En el presente capitulo se realiza un reporte final de los gastos fundamentales para el desarrollo del dispositivo propuesto basado en los costos de hardware y horas trabajadas.

El costo total de la inversión de los componentes necesarios para la implementación del hardware se detalla en la siguiente tabla.

Tabla I. **Costos del hardware**

| Concepto | Cantidad | Costo unitario | Costo |
|---|---|---|---|
| Tarjeta de desarrollo NodeMCU | 1 | Q 85,00 | Q 85,00 |
| Modulo Ultrasónico JSN-SR04T | 1 | Q 60,00 | Q 60,00 |
| Caja de acrílico 15x10x5cm | 1 | Q 80,00 | Q 80,00 |
| Cables Jumper | 5 | Q 2.00 | Q 10,00 |
| Tornillos M1*3 | 8 | Q 0.5 | Q 4,00 |
| | | **Total** | **Q 239,00** |

Fuente: elaboración propia.

Adicional a la inversión de hardware, se realiza un desglose sobre el total de horas trabajo y el costo para la implementación del dispositivo y pruebas de funcionamiento basado en el salario para personas no agrícolas establecido por el ministerio de trabajo de Guatemala que equivale a Q 11,61 por hora laboral.

Tabla II. **Costos de horas trabajadas**

| Concepto | Cantidad de horas | Costo unitario | Costo |
|---|---|---|---|
| Desarrollo del software de la tarjeta NodeMCU | 18 | Q 11,61 | Q 208,98 |
| Desarrollo del software del servidor | 30 | Q 11,61 | Q 348,30 |
| Implementación del hardware | 12 | Q 11,61 | Q 139,32 |
| Pruebas de funcionamiento | 6 | Q 11,61 | Q 69,66 |
| | | **Total** | **Q 766,26** |

Fuente: elaboración propia.

El costo final agrupando los gastos de hardware y horas trabajadas se describe en la siguiente tabla.

Tabla III. **Costo final del dispositivo**

| Concepto | Costo |
|---|---|
| Costo total de horas trabajadas | Q 239,98 |
| Costo total de componentes y hardware | Q 766,26 |
| **Total** | **Q 1 006,24** |

Fuente: elaboración propia.

# CONCLUSIONES

1. Se expusieron los conceptos fundamentales de tipos de medidores de nivel de agua potables clasificados por método de medición.

2. Mediante el uso de una tarjeta de desarrollo de placa reducida y módulos periféricos es viable una configuración de hardware de bajo costo del dispositivo de monitoreo de nivel de agua potable en cisternas.

3. El diseño de software del dispositivo depende de una serie de etapas las cuales en conjunto complementan la funcionalidad del dispositivo.

4. Se verifica el funcionamiento correcto del dispositivo para el monitoreo del nivel de agua potable en cisternas a través de una aplicación web en tiempo real mediante pruebas exitosas en cada una de sus etapas.

5. El costo final del hardware del dispositivo y de las horas trabajadas es de Q1 006,24.

# RECOMENDACIONES

1. Experimentar con diferentes tipos de medición indirecta para la toma de nivel de líquido.

2. Valorar distintos tipos de tarjetas de desarrollo para la implementación del dispositivo para mejoras de desempeño o costos.

3. Considerar los distintos tipos de librerías de diseño web para realizar mejoras de visualización gráfica en la aplicación.

4. Realizar una comparación entre la medición indirecta del sensor con un instrumento de medición directa para verificar que los resultados obtenidos sean válidos.

5. Considerar la implementación de otros lenguajes de programación para el desarrollo del servidor para mejorar la eficiencia del software.

6. Tomar en cuenta el cambio de dirección web en la tarjeta de desarrollo, al momento de implementar la aplicación en un servidor web con dominio adquirido.

7. Considerar el rango de direcciones ip asignadas en la red local para evitar conflictos de red.

8. Evaluar que se incluya el uso de materiales aislantes de humedad para el diseño del recipiente del dispositivo.

9. Propiciar que el dispositivo se encuentre cerca de la red wifi para no tener pérdidas de conexión con el servidor que almacena la información de nivel.

# BIBLIOGRAFÍA


1. COMPONENTES 101. *NodeMCU ESP8266*. [en línea]. <https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet>. [Consulta: julio 2021].

2. Concepto. *Medición*. [en línea]. <https://concepto.de/medicion/>. [Consulta: julio 2021].

3. Endress. *Medición de nivel*. [en línea]. <https://www.es.endress.com/es/instrumentacion-campo/medicion-nivel>. [Consulta: julio 2021].

4. Flask. *Quickstart*. [en línea]. <https://flask.palletsprojects.com/en/2.0.x/quickstart/>. [Consulta: julio 2021].

5. Hyperphysics. *Concepto de velocidad*. [en línea] <http://hyperphysics.phy-astr.gsu.edu/hbasees/vel2.html>. [Consulta: julio 2021].

6. Jahankitshop. *Datasheet: Módulo ultrasónico JSN-SR04*. [en línea]. <https://www.jahankitshop.com/getattach.aspx?id=4635&Type=Product>. [Consulta: julio 2021].

7. Mintrabajo. *Salario mínimo 2021 Guatemala*. [en línea]. <https://www.mintrabajo.gob.gt/index.php/dgt/salario-minimo#2020>. [Consulta: julio 2021].

# APÉNDICES

## Apéndice 1. **Código final software Arduino**

```
#include <ESP8266HTTPClient.h>
#include <ESP8266WiFi.h>
const int Trigger = D7;
const int Echo = D6;
const char *nombre_red = "WLAN";
const char *clave = "ABCDE7485";

IPAddress ip(192,168,1,200);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);

void setup()
{
  pinMode(Trigger, OUTPUT);
  pinMode(Echo, INPUT);
  digitalWrite(Trigger, LOW);
  Serial.begin(9600);                                    /
  WiFi.mode(WIFI_STA);
  WiFi.config(ip, gateway, subnet);
  WiFi.begin(nombre_red, clave);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(200);
    Serial.print('.');
  }
```

Continuación del apéndice 1.

```
  Serial.println("Conexión establecida");
  Serial.println(WiFi.localIP());
}

void loop()
{

  long t;
  long d;
  digitalWrite(Trigger, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trigger, LOW);
  t = pulseIn(Echo, HIGH);
  d = t/59;
  Serial.println(d);
  delay(100);

   HTTPClient http;
   WiFiClient client;

    http.begin(client,"http://192.168.1.8:8090/valor");
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    http.POST("dato="+String(d));
    http.end();
    http.begin(client,"http://192.168.1.8:8090/tiempo");
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    int tiempo = http.POST("tiempo=tiempo");
    http.end();
    delay(tiempo*60*1000/100);
}
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

## Apéndice 2. **Código final software Python**

```
from flask import Flask, request,render_template
from datetime import datetime
import sqlite3
app = Flask(__name__)
conector = sqlite3.connect('base_de_datos.db')
cursor = conector.cursor()
cursor.execute("CREATE TABLE IF NOT EXISTS nivel_de_agua( fecha TEXT, valor REAL)")
cursor.execute("CREATE TABLE IF NOT EXISTS valores( tiempo REAL,altura REAL,ancho RE
AL,largo REAL)")
conector.close()
@app.route('/valor', methods=['POST', 'GET'])
def obtener_valor():
    if request.method == 'POST':
        dato=request.form['dato']
        ahora = datetime.now()
        hora_fecha = ahora.strftime("%Y-%m-%d %H:%M:%S")
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("INSERT INTO nivel_de_agua VALUES('"+hora_fecha+"',"+dato+")"
)
        conector.commit()
        cursor.close()
        conector.close()
        return "Recibido"

@app.route('/tiempo', methods=['POST', 'GET'])
def tiempo():
    if request.method == 'POST':
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("SELECT * FROM valores")
        valores = cursor.fetchone()
        cursor.close()
        conector.close()
        tiempo=valores[0]
```

Continuación del apéndice 2.

```
        return str(tiempo)
@app.route('/', methods=['POST', 'GET'])
def index():
    conector = sqlite3.connect('base_de_datos.db')
    cursor = conector.cursor()
    cursor.execute( "SELECT MAX (fecha) AS 'fecha',valor FROM nivel_de_agua")
    nivel_de_agua = cursor.fetchone()
    cursor.execute("SELECT * FROM valores")
    valores = cursor.fetchone()
    altura_total=valores[1]
    ancho_total=valores[2]
    largo_total=valores[3]
    fecha=nivel_de_agua[0]
    altura=int(nivel_de_agua[1])
    nivel=int((nivel_de_agua[1]/450)*100)
    volumen_total=altura_total*ancho_total*largo_total
    volumen_nivel=altura*ancho_total*largo_total
    cursor.close()
    conector.close()
    return render_template("index.html",fecha=fecha,nivel=nivel,altura=altura,volume
n_nivel=volumen_nivel,volumen_total=volumen_total)

@app.route("/formulario",methods=["POST","GET"])
def formulario():
    if request.method == "POST":
        tiempo = request.form.get("tiempo")
        ancho = request.form.get("ancho")
        largo = request.form.get("largo")
        altura = request.form.get("altura")
        conector = sqlite3.connect('base_de_datos.db')
        cursor = conector.cursor()
        cursor.execute("DELETE FROM valores")
        conector.commit()
        cursor.execute("INSERT INTO valores VALUES("+tiempo+","+altura+","+ancho+","
+largo+")")
```

Continuación del apéndice 2.

```
        conector.commit()
        cursor.close()
        conector.close()
    return render_template('formulario.html')

app.run(host='0.0.0.0', port=8090,debug=True)
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

## Apéndice 3. **Plantilla principal - html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Visualización de nivel</title>

    <link href="{{ url_for('static', filename='main.css') }}" rel="stylesheet">
</head>
<body>
    <div class="content2">
        <div class="content">
            <div class="cuadrado">
                <div class="agua" id="agua" style="height: {{ nivel }}%;">
                    <div class="fecha">Volumen Total: {{ volumen_total }} cm^2</div>
                    <div class="fecha">Volumen Actual: {{ volumen_nivel }} cm^2</div
>
                    <div class="fecha">Altura: {{ altura }} cm</div>
                    <div class="fecha">Porcentaje: {{ nivel }}%</div>
                    <div class="fecha">Fecha: {{ fecha }}</div>
                </div>
            </div>
```

Continuación del apéndice 3.

```
            <div>
                <a href="/" class="button">Actualizar</a>
                <a href="/formulario" class="button">Configurar Valores</a>
            </div>
        </div>
    </div>
</body>
</html>
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.

## Apéndice 4. **Plantilla formulario - html**

```
<!DOCTYPE html>
<html>

<head>
    <title>Visualización de nivel</title>
    <link href="{{ url_for('static', filename='main.css') }}" rel="stylesheet">
</head>

<body>
    <div class="content2">
        <div class="content">
            <div>
                <a href="/" class="button">Volver a la aplicacion</a>
            </div>
       <form method="post" id="todo-form" action="/formulario">
       <input type="number" name="tiempo" id="tiempo" placeholder="tiempo de muestre
o [Minutos]>        <input type="number" name="altura" id="altura" placeholder="Altu
ra del tanque [Cm]">
       <input type="number" name="ancho" id="ancho" placeholder="Ancho del tanque [C
m]">
```

Continuación del apéndice 4.

```
        <input type="number" name="largo" id="largo" placeholder="Largo del tanque [C
m]">                <button type="submit" class="button_form">Enviar</button>
        </form>
         </div>
    </div>
</body>
</html>
```

Fuente: elaboración propia, empleando Visual Estudio Code 1.61.