

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA  
FACULTAD DE INGENIERÍA  
ESCUELA DE CIENCIAS Y SISTEMAS  
ARQUITECTURA DE COMPUTADORES Y  
ENSAMBLADORES 1  
ING. OTTO RENE ESCOBAR LEIVA  
JOSE MANUEL LOPEZ LEMUS



### PROYECTO #1 - CASA INTELIGENTE

GRUPO #1	
CARNET	NOMBRE COMPLETO
202300378	Maria Fernanda Morales Lima
202300712	Melvin Geovanni García Sumalá
202307378	Josue David Figueroa Acosta
202300580	Sebastian Levi Velásquez Valle
202307272	Bryan Alejandro Anona Paredes
202300370	Mariana Abigail Mejía García

GUATEMALA, 5 DE SEPTIEMBRE 2025

<b>INTRODUCCIÓN</b>	<b>3</b>
<b>OBJETIVOS</b>	<b>4</b>
General	4
Específicos	4
<b>CONTENIDO</b>	<b>5</b>
Resolución de problemas	5
Conexión de la Raspberry Pi 5 con MQTT	5
Configuración con el broker	5
Definir Tópicos	5
Creación del cliente:	6
Al conectarse exitosamente:	6
Al recibir mensajes:	6
Conexión de la Raspberry Pi 5 con MongoDB	8
Backend	9
Conexión del Backend con MongoDB	9
Conexión del Backend con MQTT	9
Endpoints con MQTT	10
Frontend	11
Diseño del login	11
Diseño de la gráfica de alarmas	11
Diseño de la gráfica de entradas	12
Diseño de la tabla de alarmas	12
Diseño del control	12
Diagrama de conexiones en Proteus	13
Equipo utilizado	13
Presupuesto	15
<b>CONCLUSIONES</b>	<b>16</b>
<b>ANEXOS</b>	<b>17</b>
Fotografías	17

# INTRODUCCIÓN

En la era digital actual, la automatización del hogar ha dejado de ser una visión futurista para convertirse en una realidad tangible y accesible. La creciente demanda de soluciones tecnológicas que mejoren la calidad de vida, optimicen el consumo energético y brinden mayor seguridad en los hogares, ha impulsado el desarrollo de sistemas domóticos cada vez más sofisticados y económicamente viables. El presente proyecto surge como respuesta a una necesidad específica planteada por la residencial "Pinos Altos", que busca transformar algunas de sus viviendas tradicionales en hogares inteligentes o "Smart Homes".

Lo que se pretende lograr con este proyecto es multifacético. En primer lugar, se busca desarrollar un prototipo funcional que demuestre la viabilidad técnica y económica de implementar sistemas domóticos utilizando hardware accesible como la Raspberry Pi, combinado con sensores y actuadores de bajo costo pero alta eficiencia. El sistema contempla la automatización de aspectos críticos del hogar como la iluminación inteligente, el control climático, la seguridad perimetral, el riego automatizado y el control de acceso.

Se busca que los estudiantes adquieran competencias en programación con Python, manejo de protocolos de comunicación IoT como MQTT, desarrollo de aplicaciones web modernas, integración con bases de datos NoSQL en la nube, y diseño de interfaces de usuario intuitivas.

# OBJETIVOS

## General

El estudiante será capaz de diseñar y desarrollar una aplicación web que permita la automatización de dispositivos electrónicos en una vivienda, integrando el uso de hardware en Raspberry Pi con el lenguaje de Python

## Específicos

- Los estudiantes serán capaces de integrar sensores y actuadores a su proyecto, automatizando funciones como encender luces o controlar la temperatura en una vivienda.
- Los estudiantes podrán diseñar y desarrollar interfaces gráficas que permitan encender o apagar dispositivos de forma remota y configurar automatizaciones sencillas.
- Los estudiantes gestionarán su proyecto de manera ágil, organizando entregables, definiendo sprints y realizando revisiones constantes de su progreso..

# CONTENIDO

## Resolución de problemas

Cuando se conectó las raspberry a las IP se presentó el problema de DNS (Domain Name System) la cual no respondió, para esto se configuró a inicios del código para usando servicios servidores publicos de DNS, por ejemplo: 8.8.8.8, 1.1.1.1; se redujeron los tiempos de timeout a 3 segundos, y lifetime a 5 segundos, esto para evitar timeout largos y que el sistema evite colgarse por estar esperando una respuesta DNS.

## Conexión de la Raspberry Pi 5 con MQTT

Para realizar la conexión de la Raspberry con MQTT se configuró de la forma siguiente:

### Configuración con el broker

```
# ===== MQTT (HiveMQ Cloud) =====
MQTT_HOST = "332362e6e921400e87b8124f8bfc0546.s1.eu.hivemq.cloud"
MQTT_PORT = 8883
MQTT_USER = "Arqui1"
MQTT_PASS = "Pass1234"
CLIENT_ID = "raspi-casainteligente-main"
```

- **Broker:** HiveMQ Cloud (servicio en la nube)
- **Seguridad:** Conexión encriptada (puerto 8883 con TLS)
- **Autenticación:** Usuario y contraseña

### Definir Tópicos

```
TOPIC_ILUM = "/ilumination"
TOPIC_ENTR = "/entrance"
TOPIC_ALERT= "/alerts"
TOPIC_VENT = "/ventilador"
TOPIC_BOMBA = "/bombaagua"
```

Creación del cliente:

```
def build_mqtt_client():
    client = mqtt.Client(
        client_id=CLIENT_ID,
        protocol=mqtt.MQTTv311,
        callback_api_version=CallbackAPIVersion.VERSION2
    )
    client.username_pw_set(MQTT_USER, MQTT_PASS)
    ctx = ssl.create_default_context()
    ctx.check_hostname = True
    ctx.verify_mode = ssl.CERT_REQUIRED
    client.tls_set_context(ctx)
    client.on_connect = on_connect
    client.on_message = on_message
    return client
```

Al conectarse exitosamente:

```
def on_connect(client, userdata, flags, reason_code, properties):
    code = getattr(reason_code, "value", reason_code)
    print(f"[MQTT] on_connect rc={code}")
    if code == 0:
        client.subscribe(TOPIC_IOLUM)
        client.subscribe(TOPIC_ENTR)
        client.subscribe(TOPIC_VENT)
        client.subscribe(TOPIC_BOMBA)
    print(f"[MQTT] Subscribed: {TOPIC_IOLUM}, {TOPIC_ENTR}, {TOPIC_VENT}, {TOPIC_BOMBA}")
```

Al recibir mensajes:

```
def on_message(client, userdata, msg):
    payload = msg.payload.decode().strip()
    top = msg.topic
    print(f"[MQTT] {top}: {payload}")
```

En este apartado se encuentran todo los tipos de mensajes de parte de todos los tópicos y retornará los que se debe de hacer dependiendo de que reciba primero .

- TOPIC\_ENTR: retorna si MQTT se encuentra abierto o cerrado (apagado o encendido), para poder realizar las demás conexiones es necesario que esté abierto.

```
# /entrada → "open"/"close" (sólo por MQTT)
if top == TOPIC_ENTR:
    if payload.lower() == "open":
        servo.max()
        col_entr.insert_one({"ts": now_iso(), "evento": "PORTON_OPEN", "origen": "mqtt"})
        lcd2("Porton:", "ABIERTO")
        time.sleep(2)
        refrescar_lcd()
    elif payload.lower() == "close":
        servo.min()
        col_entr.insert_one({"ts": now_iso(), "evento": "PORTON_CLOSE", "origen": "mqtt"})
        lcd2("Porton:", "CERRADO")
        time.sleep(2)
        refrescar_lcd()
```

- TOPIC\_VENT: en este apartado se define como la comunicación con el ventilador y se establece si está encendido o apagado.

```
# /ventilador → "on"/"off"
elif top == TOPIC_VENT:
    if payload.lower() == "on":
        set_ventilador(True, "mqtt")
        lcd2("Ventilador:", "ENCENDIDO")
        time.sleep(2)
        refrescar_lcd()
    elif payload.lower() == "off":
        set_ventilador(False, "mqtt")
        lcd2("Ventilador:", "APAGADO")
        time.sleep(2)
        refrescar_lcd()
```

- TOPIC\_BOMBA: en este apartado se define como la comunicación con la bomba y se establece si está encendido o apagado.

```
# /bombaagua → "on"/"off"
elif top == TOPIC_BOMBA:
    if payload.lower() == "on":
        set_bomba(True, "mqtt")
        col_rieg.insert_one({"ts": now_iso(), "evento": "BOMBA_ON", "origen": "mqtt"})
        lcd2("Bomba Agua:", "ENCENDIDA")
        time.sleep(2)
        refrescar_lcd()
    elif payload.lower() == "off":
        set_bomba(False, "mqtt")
        col_rieg.insert_one({"ts": now_iso(), "evento": "BOMBA_OFF", "origen": "mqtt"})
        lcd2("Bomba Agua:", "APAGADA")
        time.sleep(2)
        refrescar_lcd()
```

- TOPIC\_ILUM: para este fragmento en especial manejaremos los LEDS, dependiendo del LED y de en donde se ubique se comunicara si está encendido o apagado.

```

# /ilumination → room1 on/off | room2 on/off | rgb r,g,b
elif top == TOPIC_ILLUM:
    pl = payload.lower()
    nowdoc = {"ts": now_iso(), "origen": "mqtt"}
    if pl.startswith("room1"):
        on = "on" in pl
        GPIO.output(GPIO_Led1, GPIO.HIGH if on else GPIO.LOW)
        lcd2("Habitacion 1", "ENCENDIDA" if on else "APAGADA")
        time.sleep(2)
        refrescar_lcd()
        col_ilum.insert_one({**nowdoc, "room": "room1", "on": on})
    elif pl.startswith("room2"):
        on = "on" in pl
        GPIO.output(GPIO_Led2, GPIO.HIGH if on else GPIO.LOW)
        lcd2("Habitacion 2", "ENCENDIDA" if on else "APAGADA")
        time.sleep(2)
        refrescar_lcd()
        col_ilum.insert_one({**nowdoc, "room": "room2", "on": on})
    elif pl.startswith("room3"):
        on = "on" in pl
        GPIO.output(GPIO_Led3, GPIO.HIGH if on else GPIO.LOW)
        lcd2("Habitacion 3", "ENCENDIDA" if on else "APAGADA")
        time.sleep(2)
        refrescar_lcd()
        col_ilum.insert_one({**nowdoc, "room": "room3", "on": on})
    elif pl.startswith("rgb"):
        try:
            → vals = payload.split(None, 1) if " " in payload else payload.split(":", 1)
            r, g, b = [int(x) for x in vals.replace(" ", "").split(",")]
            set_rgb_values(r, g, b) # PWM 0..255 en cada canal
            col_ilum.insert_one({**nowdoc, "rgb": [r, g, b]}) 
            lcd2("RGB:", f"{r},{g},{b}")
            time.sleep(2)
            refrescar_lcd()
        except Exception as e:
            print("RGB payload invalido:", e)

```

## Conexión de la Raspberry Pi 5 con MongoDB

En este fragmento de código se encuentra la configuración de la conexión de MongoDB con Raspberry. En la parte superior se encuentra la conexión directa con la base de datos, mientras que en la parte inferior están las colecciones o tablas por categoría.

```

# ===== MongoDB (por categoría) =====
MONGO_URL = "mongodb+srv://Arqui1:pass123@cluster0.pij2euq.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0"
mongo = MongoClient(MONGO_URL)
db = mongo.get_database("Arqui1")

col_temp = db.get_collection("Temperatura")
col_rieg = db.get_collection("Riego")
col_ilum = db.get_collection("Iluminacion")
col_vent = db.get_collection("Ventilacion")
col_entr = db.get_collection("Entrada")
col_mov = db.get_collection("Movimiento")
col_alarm= db.get_collection("Alarmas")

```

Esto con el fin de tener almacenado el historial de todos los eventos y mediciones, y tener el acceso a la nube de mongoDB de forma remota.

## Backend

### Conexión del Backend con MongoDB

Para conectar el servicio en la nube de MongoDB al backend se realizaron varias configuraciones, comenzando con la url y se creando una instancia de “cliente” con la base de datos.

```
// ===== MongoDB =====
const uri = "mongodb+srv://Alejandro:y1zajqr9Awm4G15g@cluster0.pij2euq.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0";
const client = new MongoClient(uri);

// ===== Funciones Mongo =====
async function getAlarmas() {
  await client.connect();
  const database = client.db("Arqui1");
  const alarmas = database.collection("Alarmas");
  return alarmas.find().sort({ fecha: 1 }).limit(50).toArray();
}
```

### Conexión del Backend con MQTT

En este fragmento de código se tiene la conexión del backend con MQTT y el manejo de error cuando no se logra una conexión exitosa. Junto con la conexión de las Raspberry en el manejo de comandos como el usuario y password.

```
// ===== MQTT (HiveMQ Cloud) =====
const MQTT_HOST = "332362e6e921400e87b8124f8bfc0546.s1.eu.hivemq.cloud";
const MQTT_PORT = 8883;
const MQTT_USER = "Arqui1";
const MQTT_PASS = "Pass1234";

const mqttClient = mqtt.connect(`mqtts://${MQTT_HOST}:${MQTT_PORT}`, {
  username: MQTT_USER,
  password: MQTT_PASS,
});

mqttClient.on("connect", () => {
  console.log("✅ Backend conectado a HiveMQ");
});
mqttClient.on("error", (err) => {
  console.error("❌ Error MQTT:", err);
});
```

El endpoint de esta apartado es:

```
app.post("/api/entrada", (req, res) => {
  const { action } = req.body; // "open" | "close"
  mqttClient.publish("/entrance", action, () => {
    console.log("➡️ Portón:", action);
    res.json({ ok: true, sent: action });
  });
});
```

## Endpoints con MQTT

Cuando ya se tiene conectado la base de datos, la raspberry y MQTT, se crearan endpoints para la comunicación de los 3 con el backend, utilizando los tópicos y colecciones.

- Con la comunicación de la iluminación:

```
// Iluminación (cuartos y RGB)
app.post("/api/iluminacion", (req, res) => {
  const { room, action } = req.body;
  let payload;

  if (room === "rgb") {
    payload = `rgb ${action}`; // ej. "rgb 255,0,0"
  } else {
    payload = `${room} ${action}`; // ej. "room1 on"
  }

  mqttClient.publish("/illumination", payload, () => {
    console.log("➡️ Publicado:", payload);
    res.json({ ok: true, sent: payload });
  });
});
```

- Comunicación con la ventilación

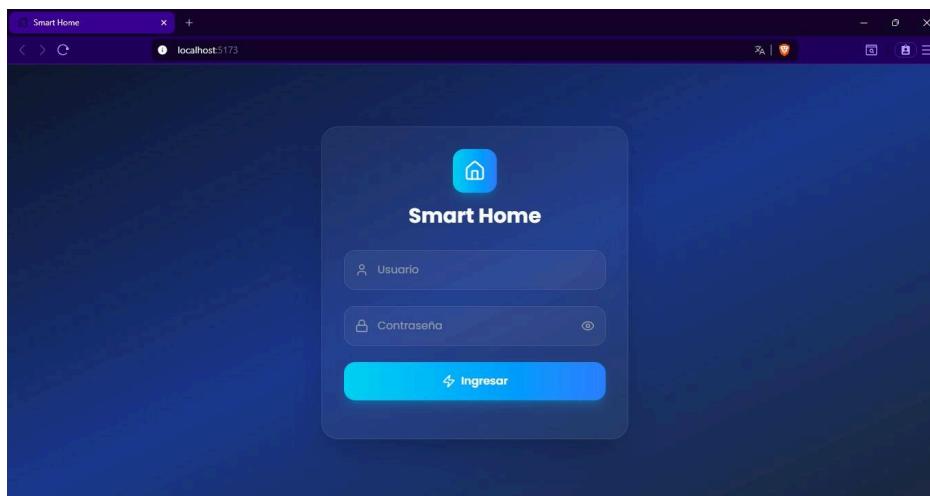
```
app.post("/api/ventilacion", (req, res) => {
  const { action } = req.body;
  mqttClient.publish("/ventilador", action, () => {
    console.log("➡️ Ventilador:", action);
    res.json({ ok: true, sent: action });
  });
});
```

- Comunicación con la bomba

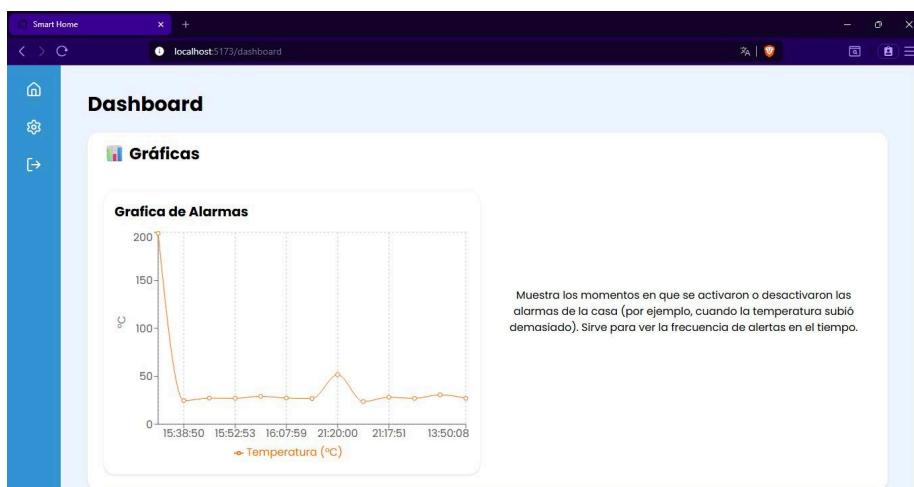
```
app.post("/api/bomba", (req, res) => {
  const { action } = req.body;
  mqttClient.publish("/bombaagua", action, () => {
    console.log("👉 Bomba:", action);
    res.json({ ok: true, sent: action });
  });
});
```

## Frontend

### Diseño del login



### Diseño de la gráfica de alarmas



## Diseño de la gráfica de entradas

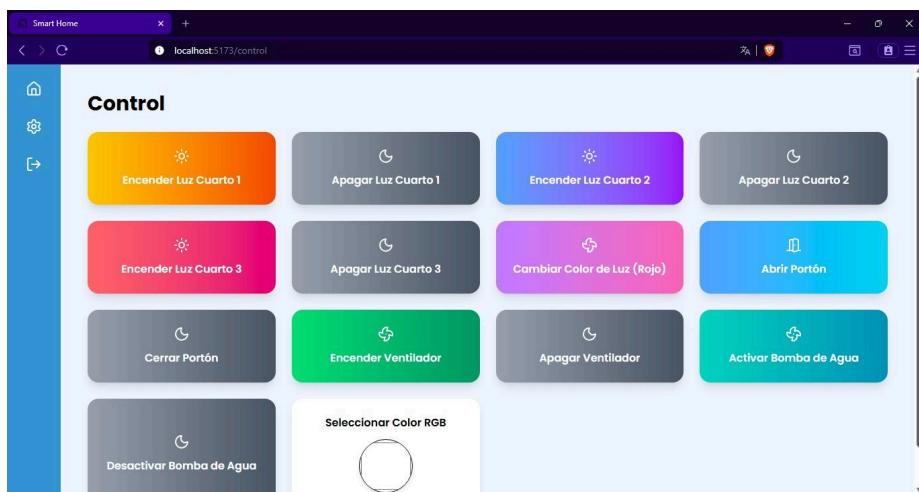


## Diseño de la tabla de alarmas

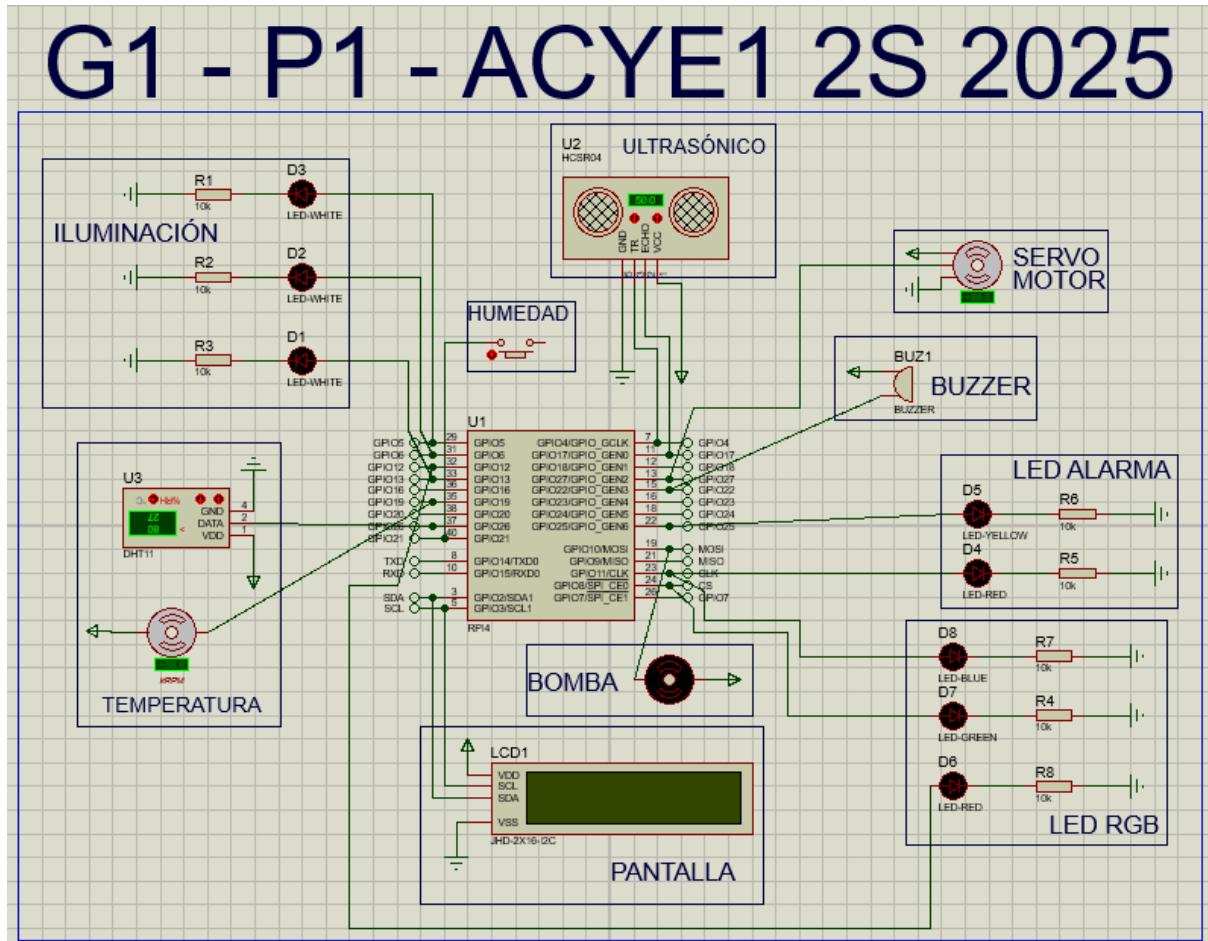
The screenshot shows a Smart Home dashboard with a sidebar containing icons for home, weather, and control. The main area displays a table titled "Tabla de Alarmas" with a filter for dates (dd/mm/aaaa). The table lists various events with their date, time, temperature, and state.

Fecha	Hora	Temperatura (°C)	Estado
27/8/2025	15:38:44	199	Alarma Activada
27/8/2025	15:38:50	24.6	Alarma Desactivada
27/8/2025	15:51:43	27.2	Alarma Activada
27/8/2025	15:52:53	27	Alarma Desactivada
27/8/2025	16:04:01	29.1	Alarma Activada
27/8/2025	16:07:59	27.3	Alarma Activada
27/8/2025	16:08:13	26.8	Alarma Desactivada
28/8/2025	21:20:00	51.6	Alarma Activada
28/8/2025	21:20:08	23.6	Alarma Desactivada
1/9/2025	21:17:51	28.1	Alarma Activada
1/9/2025	21:21:31	27	Alarma Desactivada
4/9/2025	13:47:53	30.7	Alarma Activada

## Diseño del control



## Diagrama de conexiones en Proteus



## Equipo utilizado

### Componentes Electrónicos:

- Raspberry Pi 5
- Ventiladores de 5V
- Motores de 5V
- LCD o OLED
- Leds RGB (10k)
- Leds Alarma
- Temperatura (DHT11)
- Pantalla de 16 x 12 (JHD-2X16-12C)
- Resistencias (330Ω)
- Multiplexores
- Protoboards

- Buzzer
- Jumpers
- Cargadores
- Transistores
- Servomotores
- ULTRASONICO (HCST04)
- 2 metros de manguera para pecera
- 1 hélice para motor
- Bomba de agua de 5V
- Dip switches para entrada de datos
- Cables jumper para conexiones de Q20

#### Herramientas:

- Fuente de alimentación regulada de 5V
- Multímetro digital
- Cautín y estaño
- Pinzas y alicates
- Ácido para grabado de placas
- Taladro con brocas para PCB

#### Software:

- Lenguaje de programación, se utilizó Python
- GitHub para control de documentación
- Para Base de Datos se utilizó MongoDB
- Se utilizó MQTT el cual es un protocolo de mensajería
- Para el backend se utilizó python
- Para el frontend se utilizó vite.js

## Presupuesto

Componente	Cantidad	Precio unitario	Subtotal
Resistencia IKQ 1/4W	50	Q0.50	Q25.00
Resistencia 2.2+(0 1/4W	4	Q0.50	Q2.00
Resistencia 4.3KQ 1/4W	3	Q0.50	Q1.50
Resistencia 5.1KQ 1/4W	3	Q0.50	Q1.50
Resistencia IOKQ 1/4W	1	Q0.50	Q0.50
Resistencia 20KQ 1/4W	1	Q0.50	Q0.50
LD-5WD LED blanco	6	Q1.00	Q6.00
BB612 Bomba de agua 6V	1	Q31.00	Q31.00
DHT11 sensor de humedad	1	Q20.00	Q20.00
MD-PIR2 modulo sensor 3.6 - 20V	1	Q24.00	Q24.00
MT-SG90 servo motor	1	Q28.00	Q28.00
MT-0057 motor DC 6V	1	Q6.50	Q6.50
H-175N helice par de color negro	1	Q8.00	Q8.00
H-175N helice par de color rojo	1	Q8.00	Q8.00
LD-5RGB LED catodo	3	Q2.00	Q6.00
MD-H2 modulo sensor de humedad 5V	1	Q21.00	Q21.00
2m de manguera de pecera	1	Q20.00	Q20.00
Carton	1	Q20.00	Q20.00
Placa de cobre 10)(10	2	Q12.00	Q24.00
Alambre para protoboard	8	Q2.00	Q16.00
Jumper macho	32	Q0.88	Q28.16
Broca 1/32"	4	Q2.50	Q10.00
<b>Total</b>			<b>Q307.66</b>

# CONCLUSIONES

La culminación exitosa del proyecto de Casa Inteligente para la residencial "Pinos Altos" representa un logro significativo en la aplicación práctica de tecnologías IoT. El desarrollo integral del sistema demostró que es posible crear soluciones de automatización residencial robustas y funcionales utilizando componentes de hardware accesibles. El logro principal fue la implementación exitosa de un ecosistema IoT completo que integra Raspberry Pi con sensores y actuadores diversos. El sistema logró monitorear variables ambientales como temperatura y humedad, ejecutando acciones automatizadas de control de iluminación, ventilación, riego y seguridad perimetral. La integración del protocolo MQTT estableció un enlace confiable entre los componentes físicos y la aplicación web de control.

Desde el aspecto tecnológico, el proyecto validó la efectividad de MongoDB para almacenamiento de datos IoT y la implementación de una API REST que facilita la comunicación entre la base de datos y la interfaz web, permitiendo consulta histórica y control en tiempo real.

El desarrollo de la aplicación web resultó en una interfaz intuitiva que incluye sistema de autenticación, dashboard interactivo con visualizaciones gráficas en tiempo real, y controles remotos para gestionar todos los aspectos automatizados del hogar. Las gráficas implementadas proporcionan herramientas analíticas para comprender patrones de comportamiento del sistema. El sistema de riego automatizado optimiza el uso del agua, mientras que el control automático de ventilación mantiene condiciones confortables minimizando el consumo energético.

# ANEXOS

## Fotografías

