

MANUAL DEL TRABAJADOR: SISTEMA DE DISPENSACIÓN AUTOMATIZADA CON ESP32

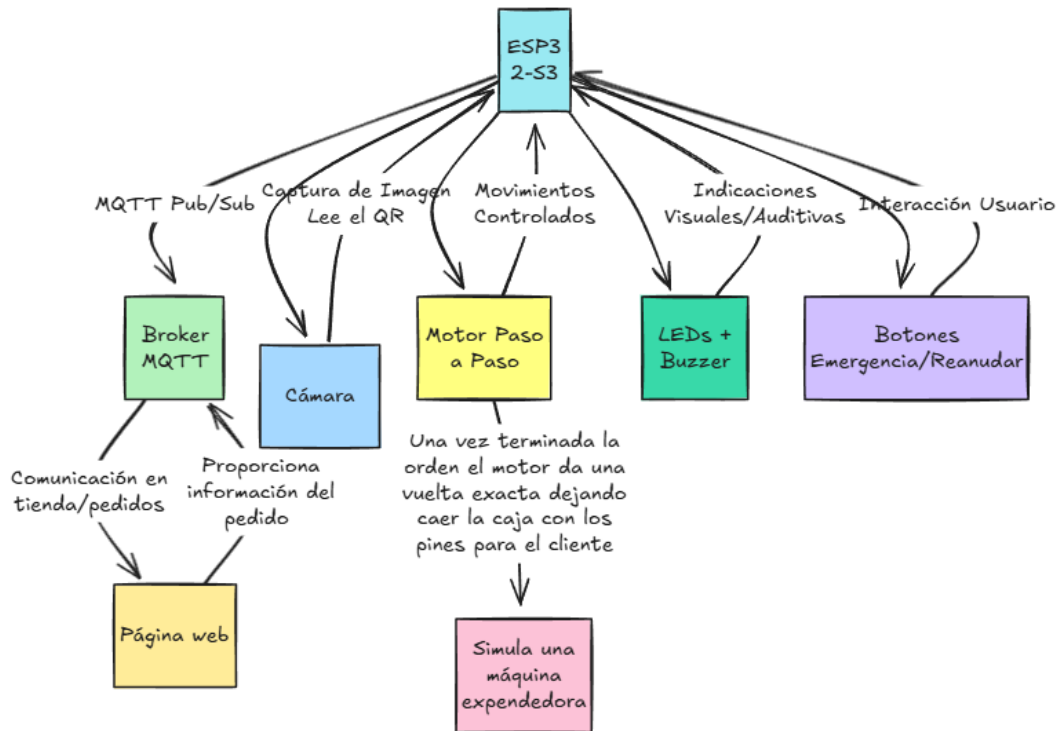
1. INTRODUCCIÓN AL SISTEMA

¡Bienvenido! Este manual te guiará paso a paso en el montaje, configuración y operación de nuestro sistema de dispensación automatizada. No necesitas ser un experto en programación, solo seguir las instrucciones cuidadosamente.

Nuestro sistema es como una pequeña máquina inteligente que puede leer **códigos QR**, procesar **pedidos** y **dispensar objetos**. Su "cerebro" es un **ESP32**, un pequeño microcontrolador que le permite hacer varias cosas a la vez, como leer la cámara y comunicarse por internet.

Este sistema está diseñado para ser:

- **Fácil de usar:** Una vez montado, su funcionamiento es muy intuitivo.
- **Fiable:** Responde rápidamente a eventos importantes, como el botón de emergencia.
- **Conectado:** Puede recibir pedidos y enviar información a otros sistemas a través de internet (MQTT).






2. HARDWARE UTILIZADO

El sistema integra diversos componentes de hardware esenciales que habilitan sus funcionalidades clave. La siguiente tabla detalla cada componente, su función, los pines GPIO asignados en el ESP32, y consideraciones específicas de su implementación.

2.1. SENSORES


Los sensores son fundamentales para la interacción del sistema con su entorno y la recopilación de información necesaria para su operación.



| Componente | Función | GPIO | Detalles | |
|------------------|------------------------------|------------------|--|---|
| Cámara | Lectura QR | GPIO4- GPIO18 | Pines conectados al sensor de imagen, configuración de clase |  |
| Botón Emergencia | Parada inmediata del sistema | GPIO14 | Configuramos el pull-up interno del ESP32 como activo. Se monta con un pull-down |  |


| Componente | Función | GPIO | Detalles | |
|----------------|----------------------|--------|--|---|
| | | | externo para asegurar una lectura fiable de la pulsación, detectando específicamente el flanco de bajada para activar la interrupción. | |
| Botón Reanudar | Reinicio del proceso | GPIO20 | Configuramos el pull-up interno del ESP32 como activo. Se monta con un pull-down externo para asegurar una lectura fiable de la pulsación, detectando específicamente el flanco de bajada para activar la interrupción. |  |

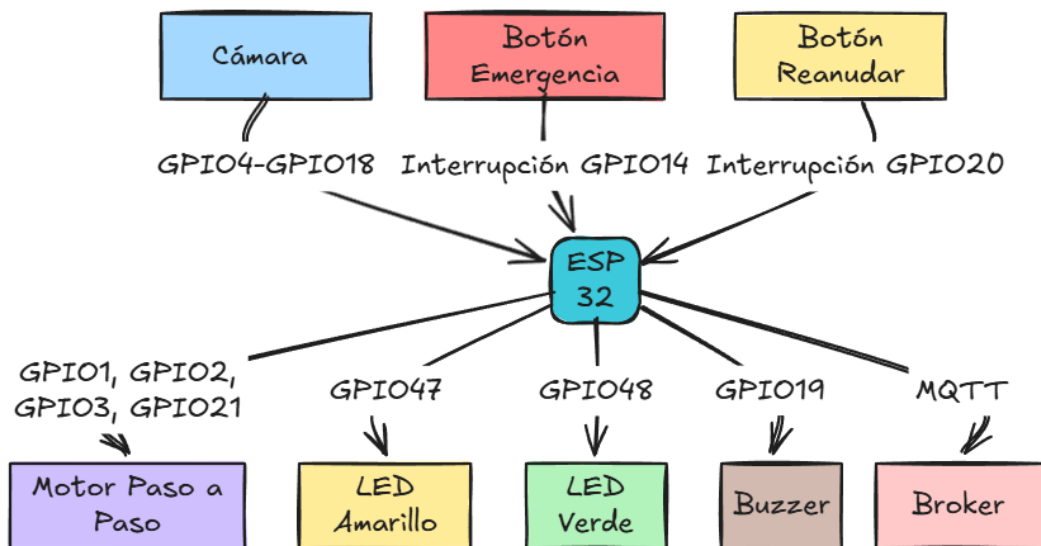
2.2. ACTUADORES

Los actuadores son responsables de las acciones físicas del sistema, transformando las decisiones del software en movimientos y señales concretas, simulando la operación de una máquina dispensadora.

| Componente | Función | GPIO | Secuencia/ Comportamiento | |
|-------------------|--|---|---|---|
| Motor Paso a Paso | Transporte de cajas de pines (simula la máquina expendedora) | IN1->GPIO1 IN2->GPIO2 IN3->GPIO3 IN4->GPIO21 | Opera en modo de onda completa , lo que requiere 2048 pasos para completar una vuelta completa (360°). La secuencia de activación de las bobinas es precisa para asegurar un |  |

| Componente | Función | GPIO | Secuencia/ Comportamiento | |
|------------------------|------------------------------------|--------|---|---|
| | | | movimiento suave y controlado que simula el mecanismo de dispensación. | |
| Fuente de alimentación | Alimentación del motor paso a paso | | Fuente externa de 5V DC, corriente mínima de 1A | |
| LED Amarillo | Indicador "en proceso" | GPIO47 | Se enciende cuando el sistema está en estado de lectura de código QR o procesando una solicitud, indicando visualmente que una operación está en curso y que el sistema está activo. |  |
| LED Verde | Indicador "éxito" | GPIO48 | Se ilumina una vez que un pedido ha sido completado y dispensado con éxito, proporcionando una confirmación visual clara al usuario o al operador. |  |

| Componente | Función | GPIO | Secuencia/ Comportamiento | |
|------------|----------------------|--------|--|---|
| Buzzer | Feedback auditivo | GPIO19 | Emite un tono de 1kHz durante 1 segundo en situaciones de emergencia (actuando como una alarma sonora) y un tono de 800Hz por 200ms para confirmar acciones como la reanudar del sistema, ofreciendo una señal auditiva distintiva para cada evento crítico. |  |



Y el circuito montado en la feria quedaría algo así, perdón por la calidad de la foto ya que es obtenida del vídeo que grabamos de la feria, debido a que si nos había pasado hacer una buena foto de todo y ya no contamos con la esp32 la cuál devolvimos.

2.3. GUÍA DE CONEXIÓN DEL CIRCUITO: ¡PASO A PASO!

Para que no te pierdas, aquí tienes una guía clara sobre cómo conectar cada componente a tu placa ESP32. **¡Asegúrate de que la ESP32 esté desconectada de la corriente mientras haces las conexiones!**

1. **Conexión de la Cámara:** la cámara ya va integrada en la esp32 así que no hace falta tocar nada
2. **Botón de Emergencia:**
 - Conecta un extremo del botón al pin **GPIO14** del ESP32.
 - Conecta el otro extremo del botón a **GND (Tierra)** del ESP32.
 - **¡Importante!** Para este botón, es recomendable usar una **resistencia pull-down** de 10k Ohm. Conecta un extremo de la resistencia entre el pin GPIO14 y un lado del botón, y el otro extremo de la resistencia a GND. Esto asegura una lectura estable.
3. **Botón de Reanudar:**
 - Conecta un extremo del botón al pin **GPIO20** del ESP32.
 - Conecta el otro extremo del botón a **GND (Tierra)** del ESP32.
 - Al igual que el botón de emergencia, se recomienda una **resistencia pull-down** de 10k Ohm. Conecta un extremo de la resistencia entre el pin GPIO20 y un lado del botón, y el otro extremo de la resistencia a GND.
4. **LED Amarillo (Indicador "en proceso"):**
 - Conecta la **pata larga (ánodo)** del LED al pin **GPIO47** del ESP32, a través de una **resistencia de 220 Ohm** (para proteger el LED).
 - Conecta la **pata corta (cátodo)** del LED a **GND (Tierra)** del ESP32.
5. **LED Verde (Indicador "éxito"):**
 - Conecta la **pata larga (ánodo)** del LED al pin **GPIO48** del ESP32, a través de una **resistencia de 220 Ohm**.
 - Conecta la **pata corta (cátodo)** del LED a **GND (Tierra)** del ESP32.
6. **Buzzer:**
 - Conecta el **pin positivo (+) o el pin de señal** del buzzer al pin **GPIO19** del ESP32.

- Conecta el **pin negativo (-) o GND** del buzzer a **GND (Tierra)** del ESP32.

7. Motor Paso a Paso (y su Driver ULN2003):

- **Conexión del motor al driver ULN2003:**
 - Tu motor paso a paso tiene 4 o 5 cables. Estos cables se conectan a las **entradas numeradas (IN1, IN2, IN3, IN4)** del driver ULN2003. Sigue el orden de cables de tu motor para conectarlos correctamente (a menudo vienen etiquetados o se encuentran en el datasheet del motor).
- **Conexión del driver ULN2003 al ESP32:**
 - El driver ULN2003 tiene 4 pines de entrada que se conectan al ESP32:
 - **IN1** del driver a un pin GPIO1 del ESP32
 - **IN2** del driver a otro pin GPIO2 del ESP32.
 - **IN3** del driver a otro pin GPIO3 del ESP32.
 - **IN4** del driver a otro pin GPIO21 del ESP32.
 - Conecta el pin **GND** del driver ULN2003 a **GND (Tierra)** del ESP32.
- **Alimentación del Driver y el Motor:**
 - Conecta el pin **VCC (o +5V)** del driver ULN2003 al **positivo (+) de tu fuente de alimentación externa de 5V DC**.
 - Conecta el pin **GND** del driver ULN2003 al **negativo (-) de tu fuente de alimentación externa de 5V DC**.
 - **¡Importante!** Asegúrate de que la **tierra (GND) del ESP32 y la tierra (GND) de la fuente de alimentación externa estén conectadas entre sí**. Esto es crucial para que compartan una referencia común.

3. PREPARANDO TU ENTORNO DE TRABAJO (SOFTWARE)

Para que el ESP32 funcione, necesitamos cargarle un programa. Usaremos **Arduino IDE**, una herramienta sencilla y muy popular.

3.1. DESCARGAR E INSTALAR ARDUINO IDE

1. **Ve a la página oficial:** Abre tu navegador y visita www.arduino.cc/downloads.
2. **Elige tu sistema operativo:** Busca la versión que corresponda a tu computadora (Windows, macOS, Linux).
3. **Descarga el instalador:** Haz clic en el enlace de descarga. Puedes elegir "Just Download" si no quieres contribuir.
4. **Instala Arduino IDE:**
 - **Windows:** Ejecuta el archivo .exe descargado y sigue las instrucciones. Acepta los controladores que te pida instalar.
 - **macOS:** Arrastra el ícono de Arduino al directorio de Aplicaciones.
 - **Linux:** Descomprime el archivo y sigue las instrucciones del archivo install.sh.

3.2. CONFIGURAR ARDUINO IDE PARA ESP32

Para que Arduino IDE pueda programar el ESP32, necesitamos añadirle sus "instrucciones especiales":

1. **Abre Arduino IDE.**
2. **Ve a Preferencias:**
 - En Windows: Archivo > Preferencias
 - En macOS: Arduino IDE > Preferencias
3. **Instala el paquete ESP32:**
 - Ve a Herramientas > Placa > Gestor de Tarjetas...

- En la barra de búsqueda, escribe "**esp32**".
- Busca la entrada "esp32 by Espressif Systems" y haz clic en "**Instalar**". Espera a que termine la instalación.

3.3. INSTALAR LIBRERÍAS ADICIONALES

El programa que vamos a usar necesita algunas librerías extras para funcionar correctamente. ¡No te preocupes, es sencillo!

1. Librerías que ya vienen con Arduino IDE o el paquete ESP32 (no necesitan instalación manual):

- Arduino.h
- WiFi.h
- freertos/queue.h
- freertos/task.h

2. Librerías que necesitas instalar desde el Gestor de Librerías de Arduino:

- **PubSubClient**: Esta librería permite la comunicación MQTT.
 - Ve a Programa > Incluir Librería > Gestionar Librerías...
 - En la barra de búsqueda, escribe "**PubSubClient**".
 - Busca la librería "**PubSubClient by Nick O'Leary**" y haz clic en "**Instalar**".
- **ArduinoJson**: Esta librería facilita el manejo de mensajes en formato JSON.
 - Repite los pasos anteriores: Programa > Incluir Librería > Gestionar Librerías...
 - En la barra de búsqueda, escribe "**ArduinoJson**".
 - Busca la librería "**ArduinoJson by Benoît Blanchon**" y haz clic en "**Instalar**".

3. Librería quirc (Puede requerir instalación manual si no está incluida en el paquete de la cámara o ESP32):

La librería src/quirc/quirc.h a menudo viene con el soporte de la cámara para ESP32 o ya está incluida en la instalación del paquete esp32. Si al compilar te da un error relacionado con quirc.h, es posible que necesites ubicar la ruta correcta en tu sistema o, en casos muy específicos, descargarla e incluirla manualmente en la carpeta

src/quirk dentro de tu *sketch*. Para la mayoría de los casos de uso con `esp_camera.h`, ya debería estar disponible.

3.4. SELECCIONAR TU PLACA ESP32

1. **Conecta tu ESP32 al ordenador** usando un cable USB.
2. En Arduino IDE, ve a Herramientas > Placa > ESP32 Arduino.
3. Busca tu modelo específico de ESP32. Si tienes la placa **"ESP32-S3-DevKitC-1"** o **"ESP32-S3-Eye"** (si tu cámara es parte de la placa), selecciónala. Si no estás seguro, prueba con **"ESP32 Dev Module"** como opción genérica.
4. **Selecciona el Puerto USB:** Ve a Herramientas > Puerto. Verás una lista de puertos seriales. Elige el que corresponde a tu ESP32. (En Windows, suele ser "COMx"; en macOS/Linux, "/dev/cu.usbserial-xxxx" o similar).

3.5. CARGAR EL PROGRAMA (FIRMWARE)

Ahora que tienes tu entorno listo, es hora de cargar el "cerebro" al ESP32:

1. **Abre el archivo del programa (sketch):** Tu programa (firmware) debería estar en un archivo .ino (ej. `mi_proyecto_esp32.ino`). Ábrelo con Arduino IDE.
2. **Configura tus credenciales Wi-Fi:** Dentro del código, busca las líneas donde se definen el **nombre de tu red Wi-Fi (SSID)** y la **contraseña (password)**. Modifícalas para que coincidan con tu red.

```
const char* ssid = "TU_NOMBRE_DE_WIFI";  
const char* password = "TU_CONTRASENA_DE_WIFI";
```

3. **Compilar (Verificar):** Haz clic en el botón de la **palomita (✓)** en la barra de herramientas de Arduino IDE. Esto verifica que el código no tenga errores. Si todo va bien, verás "Compilación terminada" en la parte inferior.
4. **Subir (Cargar):** Haz clic en el botón de la **flecha (→)** hacia la derecha. Esto cargará el programa compilado a tu ESP32. Durante este proceso, verás mensajes en la parte inferior y, a veces, un LED parpadeando en tu ESP32. **Puede que tengas que presionar y mantener el botón "BOOT" (o "FLASH") en tu ESP32 mientras lo conectas por USB, y luego soltarlo, si la carga**

falla. Si no te sale ningún error y ves "Carga terminada", ¡felicidades, el programa está en tu ESP32!

4. FUNCIONAMIENTO DEL SISTEMA: EL "CEREBRO" EN ACCIÓN

Una vez que el programa está cargado, el sistema empieza a trabajar. Aquí te explicamos cómo funciona:

4.1. INICIO DEL SISTEMA Y CONEXIÓN

Cuando enciendes el ESP32, hace lo siguiente:

1. **Prepara sus componentes:** Configura todos los pines para los LEDs, botones y el motor. Asegura que los LEDs estén apagados y el motor detenido.
2. **Prepara la comunicación:** Inicia la comunicación con el ordenador (para depuración) y configura los botones de emergencia y reanudar para que respondan **instantáneamente** a tus pulsaciones (usando "interrupciones").
3. **Prepara la "fila de espera":** Crea un sistema especial llamado "mutex" para proteger la "cola de pedidos", que es donde se guardan los pedidos que tiene que procesar.
4. **Conecta al Wi-Fi:** Busca y se conecta a tu red Wi-Fi. Esto es crucial para que pueda comunicarse con otros sistemas.
5. **Se conecta a la "central de pedidos":** Una vez en Wi-Fi, se conecta a un servidor especial en internet (llamado *broker* MQTT). Aquí, se "suscribe" a unos canales específicos para **recibir nuevos pedidos** y **enviar el estado** de las operaciones.
6. **Activa la cámara:** Configura la cámara para que pueda leer códigos QR de manera eficiente, enfocándose en un tamaño de imagen y color adecuados para la lectura.
7. **Arranca el "lector de QR":** Inicia una tarea independiente que se dedicará solo a buscar y leer códigos QR con la cámara.

4.2. CICLO DE OPERACIÓN PRINCIPAL

El sistema principal opera en un ciclo constante, como un director de orquesta que coordina todo:

- **Siempre conectado:** Mantiene la conexión con la "central de pedidos" (MQTT) para recibir y enviar mensajes.
- **Revisa pedidos:** Si no está esperando un QR y hay pedidos pendientes en su "fila de espera", toma el siguiente y empieza a procesarlo.
- **Procesa QR leído:** Si el lector de QR ha detectado y decodificado un código QR, el sistema envía la información actualizada del pedido a la "central de pedidos" y luego se prepara para el siguiente ciclo.
- **Pausa mínima:** Hace una pequeña pausa para asegurar que todas las partes del sistema tengan tiempo de funcionar correctamente.

4.3. FLUJO DE LECTURA Y PROCESAMIENTO DE QR

La cámara y su tarea asociada funcionan de forma inteligente:

- **Activo solo cuando es necesario:** El lector de QR solo activa la cámara y busca códigos cuando el sistema se lo indica (cuando está "esperando un QR"). Esto ahorra energía y recursos.
- **Captura y decodifica:** Toma una foto, la procesa y, si encuentra un QR, lo decodifica.
- **Reporta el resultado:** Si la lectura es exitosa, guarda el contenido del QR y le indica al sistema principal que un QR ha sido leído.
- **Libera recursos:** Después de cada intento, libera la memoria de la cámara para que todo siga funcionando sin problemas.
- **Ritmo inteligente:** Escanea más rápido cuando busca un QR (cada 100 milisegundos) y más lento cuando está inactivo (cada 500 milisegundos) para optimizar el rendimiento.

4.4. MANEJO DE MENSAJES MQTT (LA "CENTRAL DE PEDIDOS")

Cuando el sistema recibe un mensaje de la "central de pedidos" (MQTT), reacciona así:

- **Mensajes de respuesta (tienda/respuesta):** Si recibe un mensaje con "estado": "fin", significa que un pedido debe ser completado. Entonces, el sistema:
 - Apaga el LED amarillo y enciende el LED verde (¡listo!).
 - Activa el motor para simular la dispensación del producto.
 - Espera unos segundos con el LED verde encendido y luego lo apaga.
- **Mensajes de nuevos pedidos (tienda/pedidos):** Si recibe un nuevo pedido con el "tipo": "no" (indicando que aún falta el QR), lo añade a su "fila de espera" para procesarlo más tarde.

4.5. CONTROL DE ACTUADORES (LAS "ACCIONES FÍSICAS")

Los actuadores son los encargados de **traducir las decisiones del software en acciones físicas**:

- **Motor Paso a Paso:** Este motor simula el movimiento de una máquina expendedora. Se le envía una secuencia precisa de pulsos para que gire suavemente y dispense un "producto" (por ejemplo, una caja de pines). También puede detenerse de inmediato si hay una emergencia.
- **LEDs y Buzzer:** Son como las luces y sonidos de la máquina.
 - **LED Amarillo:** Se enciende cuando el sistema está ocupado (leyendo QR o procesando un pedido).
 - **LED Verde:** Se enciende cuando un pedido se ha completado con éxito.
 - **Buzzer:** Emite un tono fuerte para alertar en caso de emergencia y un tono diferente para confirmar que el sistema ha vuelto a funcionar.

5. CÓMO VERIFICAR QUE TODO FUNCIONA CORRECTAMENTE

Una vez que hayas montado el circuito y cargado el programa, puedes verificar el funcionamiento de la siguiente manera:

1. **Abre el Monitor Serie en Arduino IDE:** Ve a Herramientas > Monitor Serial. Asegúrate de que la velocidad (baudios) esté configurada a 115200. Aquí verás los mensajes que el ESP32 te envía sobre lo que está haciendo.

2. **Verifica la conexión Wi-Fi y MQTT:**

- Al inicio, deberías ver mensajes en el Monitor Serie que indican que el ESP32 está intentando conectarse al Wi-Fi.
- Una vez conectado al Wi-Fi, debería intentar conectarse al *broker* MQTT. Verás mensajes como "Conectado al broker MQTT".

3. **Envía un pedido de prueba (usando MQTT):**

- Necesitarás una herramienta para enviar mensajes MQTT, como **MQTT Explorer** (descárgalo e instálalo si no lo tienes) o un cliente MQTT online.
- Conéctate al mismo *broker* MQTT (broker.emqx.io) que tu ESP32.
- **Publica un mensaje en el tópico tienda/pedidos** con el siguiente formato JSON:

```
{ "talla": "M", "color": "azul", "tipo": "no" }
```

○

Observa el sistema:

- El **LED amarillo** debería encenderse.
- En el Monitor Serie, verás mensajes indicando que se ha recibido el pedido y que el sistema está esperando un QR.

4. **Muestra un Código QR a la cámara:**

- Prepara un código QR que contenga un texto simple (por ejemplo, "producto_A" o "dispensar_1").
- Acerca el código QR a la cámara del ESP32.
- **Observa el sistema:**
 - En el Monitor Serie, deberías ver mensajes como "QR decodificado correctamente" y el contenido del QR.
 - El **motor paso a paso** debería girar, simulando la dispensación.
 - El **LED amarillo** debería apagarse y el **LED verde** encenderse por unos segundos, luego apagarse.

- En tu cliente MQTT, deberías ver un mensaje publicado en tienda/pedidos con el campo tipo actualizado con el contenido de tu QR (ej. {"color": "azul", "talla": "M", "estado": "compra", "tipo": "producto_A"}).

5. Prueba el botón de emergencia:

- Pulsa el **botón de emergencia**.
- **Observa el sistema:**
 - El **motor paso a paso** debería detenerse inmediatamente.
 - El **buzzer** debería emitir un tono de alarma (1kHz por 1 segundo).
 - En el Monitor Serie, verás un mensaje de "¡Emergencia activada!".
 - En tu cliente MQTT, deberías ver una notificación de "emergencia" publicada.

6. Prueba el botón de reanudar:

- Después de una emergencia, pulsa el **botón de reanudar**.
- **Observa el sistema:**
 - El **buzzer** debería emitir un tono de confirmación (800Hz por 200ms).
 - Los LEDs deberían parpadear brevemente (LED verde dos veces).
 - En el Monitor Serie, verás un mensaje de "Sistema reanudado".
 - En tu cliente MQTT, deberías ver una notificación de "reanudar" publicada.

6. DOCUMENTACIÓN ADICIONAL DEL ESP32

Para ayudarte a entender mejor el "cerebro" de nuestro sistema y qué hace el programa que le hemos cargado, aquí tienes algunos detalles extra.

6.1. PINES DEL ESP32: ¿DÓNDE CONECTAR CADA CABLE?

Tu **ESP32** tiene muchas "patitas" o **pines**, y cada una sirve para algo diferente. Aquí te explicamos lo básico para que sepas dónde conectar cada componente.

- **Diagrama de Pines (Pinout):** Sería ideal que pegaras aquí una imagen del pinout de tu modelo específico de ESP32 (por ejemplo, "ESP32-S3-Eye" o "ESP32-S3-DevKitC-1"). Puedes buscarla en Google como "ESP32-S3-Eye pinout" y te aparecerán muchas imágenes claras.
 - *Aquí puedes insertar la imagen del pinout de tu ESP32.*
- **Tipos de Pines:**
 - La mayoría de los pines son **GPIO** (General Purpose Input/Output), que significa "Entrada/Salida de Propósito General". Son los más comunes y los usamos para conectar botones (reciben una señal) o LEDs (envían una señal).
 - Algunos pines tienen funciones especiales o están reservados, como los que se usan para la **cámara** o para la comunicación con tu ordenador. Siempre es importante respetar los números de pin indicados en la tabla de hardware para que todo funcione.

6.2. EL CÓDIGO (SKETCH) EN ARDUINO IDE: UNA MIRADA RÁPIDA

Aunque no necesitas ser programador, entender la estructura del código te puede ayudar a ubicarte.

- **setup(): La Preparación Inicial**
 - Esta parte del código se ejecuta **solo una vez** cuando el ESP32 se enciende o se reinicia. Es como el "arranque" del sistema.
 - Aquí es donde el programa configura los pines (si son de entrada o salida), se conecta a tu red Wi-Fi y establece la conexión con el *broker* MQTT. Piensa en ello como la lista de tareas que el ESP32 debe hacer antes de empezar a trabajar.
- **loop(): El Corazón del Sistema**
 - Después de que setup() termina, la función loop() se ejecuta **repetidamente, una y otra vez, sin parar**.

- Aquí es donde el ESP32 revisa constantemente si hay nuevos mensajes MQTT, si hay pedidos pendientes en la cola, o si el lector de QR ha terminado su trabajo. Es el ciclo principal donde el sistema está siempre "escuchando" y "actuando".
- **Comentarios en el Código (// o /* */):**
 - Dentro del programa, verás líneas que empiezan con // o bloques de texto entre /* y */. ¡Estas son **notas para nosotros!**
 - Los programadores usamos estos **comentarios** para explicar qué hace cada parte del código. Si alguna vez necesitas revisar algo, como cambiar las credenciales Wi-Fi, busca estos comentarios; te guiarán por el código.

6.3. RESOLUCIÓN DE PROBLEMAS COMUNES (TROUBLESHOOTING)

Es normal que surjan pequeños inconvenientes. Aquí tienes algunas soluciones para los problemas más habituales.

- **Problema: El ESP32 no aparece en Herramientas > Puerto en Arduino IDE.**
 - **Solución:**
 - Asegúrate de que estás usando un **cable USB de datos**, no solo de carga. Algunos cables USB solo transfieren energía y no datos.
 - Podrías necesitar instalar un **controlador (driver)** para el chip USB de tu ESP32. Los más comunes son el **CP210x** o el **CH340**. Busca en Google "driver CP210x ESP32" o "driver CH340 ESP32" y sigue las instrucciones para instalarlos en tu ordenador.
 - Intenta conectar el ESP32 a **otro puerto USB** de tu ordenador.
- **Problema: El programa no se carga al ESP32 (errores al "Subir").**
 - **Solución:**
 - Verifica que has seleccionado la **placa correcta** en Herramientas > Placa > ESP32 Arduino (por ejemplo, "ESP32-S3-Eye" o "ESP32 Dev Module").
 - Asegúrate de que el **puerto USB** seleccionado sea el correcto para tu ESP32 en Herramientas > Puerto.

- En algunos modelos de ESP32, necesitas presionar y **mantener el botón "BOOT" (o "FLASH")** en la placa mientras haces clic en "Subir" en Arduino IDE. Una vez que empiece a cargar el programa, puedes soltar el botón.
- **Problema: El sistema no se conecta al Wi-Fi.**
 - **Solución:**
 - Revisa cuidadosamente las **credenciales de Wi-Fi** en el código (ssid y password). ¡Son sensibles a mayúsculas, minúsculas y espacios! Cualquier error aquí impedirá la conexión.
 - Asegúrate de que tu ESP32 esté lo suficientemente **cerca del router Wi-Fi** para tener una buena señal.
 - Verifica que la red Wi-Fi que intentas usar sea de **2.4 GHz**, ya que la mayoría de los ESP32 no son compatibles con redes de 5 GHz.
- **Problema: El sistema se conecta al Wi-Fi, pero no al *broker* MQTT.**
 - **Solución:**
 - Confirma que la **dirección del *broker* MQTT** (por ejemplo, broker.emqx.io) y el **puerto** (normalmente 1883) estén correctos en el código.
 - Si estás en una red corporativa o con un firewall estricto, es posible que el puerto 1883 esté bloqueado. Prueba a usar tu móvil como **punto de acceso Wi-Fi** para descartar problemas de red.
- **Problema: El Monitor Serial muestra mensajes de error extraños o el sistema se reinicia solo.**
 - **Solución:**
 - Esto puede ser un problema de **memoria insuficiente** o de un error de software. Asegúrate de que las librerías estén bien instaladas y que la placa seleccionada sea la correcta.
 - Si ves mensajes como "Guru Meditation Error", significa que algo ha ido muy mal en el código. A menudo, esto está relacionado con accesos incorrectos a la memoria. En estos casos, a veces ayuda reiniciar el ESP32 o volver a cargar el programa.

6.4. RECURSOS ADICIONALES PARA APRENDER MÁS

Si tienes curiosidad o necesitas profundizar en algún tema, aquí te dejamos algunos puntos de partida:

- **Documentación de Espressif:** El fabricante del ESP32, Espressif, tiene mucha documentación técnica en su sitio web. Es más avanzada, pero muy completa.
- **Comunidades de Arduino y ESP32:** Existen foros y grupos online enormes donde la gente comparte proyectos y ayuda a resolver dudas. Busca "Foro Arduino español" o "ESP32 community" para encontrar ayuda.
- **Canales de YouTube y Blogs:** Hay muchos tutoriales en video y artículos en blogs que explican cómo trabajar con el ESP32 y FreeRTOS de forma muy visual.

7. ENTENDIENDO EL CÓDIGO DEL SISTEMA (EL FIRMWARE)

La idea de este apartado es dar una visión general del código que se carga en el ESP32. No necesitas ser un programador, pero entender qué hace cada sección te ayudará a comprender cómo funciona el sistema y a localizar configuraciones importantes.

El código está escrito en un lenguaje similar a C++, usando la estructura de Arduino IDE, que es muy popular para microcontroladores.

7.1. INCLUIR LIBRERÍAS (#INCLUDE)

Al principio del código, verás varias líneas que comienzan con #include. Estas líneas le dicen al programa qué "cajas de herramientas" externas necesita para funcionar. Cada librería añade funciones y características específicas:

- **<Arduino.h>:** La librería básica de Arduino que permite usar funciones como digitalWrite(), pinMode(), delay(), etc.
- **"src/quirc/quirc.h":** Es la librería principal para **decodificar códigos QR**. Es la que hace el trabajo pesado de entender la imagen de la cámara.

- **"esp_camera.h"**: La librería que controla el **módulo de la cámara** (como la ESP32-CAM o la cámara integrada en tu placa ESP32-S3-Eye). Permite tomar fotos y configurarla.
- **<WiFi.h>**: Permite que el ESP32 se **conecte a una red Wi-Fi**.
- **<PubSubClient.h>**: Esta es la librería clave para la **comunicación MQTT**. Permite que el ESP32 envíe y reciba mensajes de la "central de pedidos".
- **<ArduinoJson.h>**: Facilita el **manejo de datos en formato JSON**. El sistema usa JSON para enviar y recibir pedidos de manera estructurada.
- **<freertos/queue.h> y <freertos/task.h>**: Estas son librerías del sistema operativo **FreeRTOS** que corre dentro del ESP32. Permiten crear "tareas" (programas que corren al mismo tiempo) y "colas" (para organizar los pedidos) de manera eficiente.

7.2. CONFIGURACIÓN GLOBAL Y VARIABLES IMPORTANTES

Aquí se definen los parámetros principales del sistema y las "cajas de información" (variables) que el programa usará en todas sus partes.

- **Credenciales Wi-Fi (ssid, password):**

```
const char* ssid = "TU_NOMBRE_DE_WIFI";
const char* password = "TU_CONTRASEÑA_DE_WIFI";
```

¡Aquí es donde debes cambiar el nombre y la contraseña de tu red Wi-Fi!

- **Configuración MQTT (mqtt_broker, topic_pedidos, topic_respuesta, mqtt_port):**

```
const char *mqtt_broker = "broker.emqx.io"; // Dirección del servidor MQTT
const char *topic_pedidos = "tienda/pedidos"; // Canal para enviar/recibir pe
const char *topic_respuesta = "tienda/respuesta"; // Canal para recibir respu
const int mqtt_port = 1883; // Puerto de conexión
```

- **Pines de los Componentes (#define):**

```
// Tamaño de la cola circular
#define QUEUE_SIZE 10
#define LED_AMARILLO 47 // GPIO14 para LED amarillo
#define LED_VERDE 48 // GPIO15 para LED verde
#define BOTON_EMERGENCIA 14 // GPIO14 para botón de emergencia (pull-up interno)
#define BOTON_REANUDAR 20 // GPIO20 para botón de reanudar (pull-up interno)
#define BUZZER_PIN 19 // GPIO19 para el buzzer
#define IN1 1
#define IN2 2
#define IN3 3
#define IN4 21
```

Estas líneas asignan un nombre fácil de recordar a cada número de pin GPIO del ESP32 donde conectamos nuestros LEDs, botones, buzzer y el driver del motor.

- **Cola de Pedidos (struct Pedido, colaPedidos, frente, final, contador):** Esta sección define cómo el sistema guarda y gestiona los pedidos que recibe. La colaPedidos es como una fila donde los pedidos esperan ser procesados. frente indica el primer pedido, final dónde añadir el siguiente, y contador cuántos pedidos hay.
- **Variables de Control (esperandoQR, qrLeido, QRCodeResult, xMutex):** Estas variables ayudan al sistema a saber en qué estado se encuentra (ej. si está esperando un QR, si ya lo leyó) y a coordinar las tareas para evitar problemas (el xMutex asegura que dos tareas no intenten modificar la cola al mismo tiempo).
- **Configuración de Pines de la Cámara (#define PWDN_GPIO_NUM, etc.):** Aquí se especifican los pines exactos a los que la cámara está conectada en la ESP32. Estos valores son muy específicos del modelo de tu cámara y ESP32.
- **Secuencias del Motor (pasosAntihorario):** Define la secuencia de señales eléctricas que el motor paso a paso necesita para girar en un sentido específico.

7.3. FUNCIONES DE AYUDA (COMPORTAMIENTOS ESPECÍFICOS)

Estas son "mini-programas" o "recetas" que el ESP32 ejecuta para hacer cosas específicas.

- **manejarEmergencia() y manejarReanudar():** Estas funciones se activan cuando pulsas el botón de emergencia o el de reanudar. Se encargan de detener el motor, activar el buzzer, enviar mensajes MQTT y parpadear los LEDs.

- **Funciones de Cola (encolarPedido, desencolarPedido, obtenerProximoPedido, hayPedidosPendientes):** Controlan la "fila de espera" de pedidos. Permiten añadir nuevos pedidos, sacar los que ya se han procesado, y verificar si hay pedidos pendientes.
- **callback() (Recepción de Mensajes MQTT):** Esta es una de las funciones más importantes. Se activa **automáticamente cada vez que el ESP32 recibe un mensaje MQTT** en los tópicos a los que está suscrito (tienda/pedidos o tienda/respuesta). Analiza el mensaje (que viene en formato JSON) y decide qué acción tomar:
 - Si el mensaje es de tienda/respuesta y dice "estado": "fin", ¡significa que el pedido está completo! El ESP32 apaga el LED amarillo, enciende el verde y activa el motor para dispensar el producto.
 - Si el mensaje es de tienda/pedidos y el "tipo" es "no", es un nuevo pedido pendiente de QR. Lo añade a la "cola de pedidos".
- **Funciones del Motor (girarUnaVueltaAntihorario, ejecutarPaso, detenerMotor):** Controlan el movimiento del motor paso a paso. girarUnaVueltaAntihorario() hace que el motor gire una vuelta completa, ejecutarPaso() envía una señal individual al motor y detenerMotor() apaga las señales para que el motor se quede quieto.
- **procesarPedido():** Esta función se encarga de iniciar el proceso de un pedido pendiente. Enciende el LED amarillo y le indica al sistema que debe empezar a esperar un código QR.
- **QRCodeReader() (Tarea del Lector QR):** Esta es una **"tarea" independiente** que funciona en paralelo con el resto del programa (gracias a FreeRTOS). Su único trabajo es:
 - Esperar la señal de que el sistema necesita un QR (esperandoQR = true).
 - Cuando la recibe, activa la cámara, toma una foto.
 - Intenta encontrar y decodificar un código QR en la imagen.
 - Si encuentra y decodifica un QR, guarda el resultado y le indica al sistema principal (qrLeido = true).
 - Libera la memoria de la cámara para que pueda tomar otra foto.
 - Hace una pausa para no consumir todos los recursos.

- **enviarPedidoCompleto():** Cuando un código QR ha sido leído, esta función toma el pedido que estaba esperando el QR, le añade la información del QR leído (actualizando el campo "tipo" del JSON) y luego lo **publica de vuelta en el tópico tienda/pedidos** en MQTT. También marca el pedido como procesado en la cola.

7.4. LAS FUNCIONES PRINCIPALES: SETUP() Y LOOP()

Estas son las dos funciones que le dan vida al programa:

- **setup(): La Configuración Inicial**
 - Se ejecuta **una sola vez** al encender el ESP32.
 - **Configura Pines:** Establece si cada pin es una entrada (para botones) o una salida (para LEDs, buzzer, motor). También asegura que los LEDs y el motor estén apagados al inicio.
 - **Inicia el Monitor Serial:** Prepara la comunicación con tu ordenador para que puedas ver los mensajes de depuración.
 - **Configura Interrupciones:** Esto es muy importante para los botones de emergencia y reanudar. Significa que el ESP32 estará "escuchando" constantemente estos pines, y si se pulsan, activará las funciones `handleEmergencia()` o `handleReanudar()` **de inmediato**, sin importar lo que esté haciendo en ese momento.
 - **Crea el Mutex:** Prepara el sistema para proteger la cola de pedidos.
 - **Conexión Wi-Fi:** Intenta conectar el ESP32 a la red Wi-Fi configurada.
 - **Configuración y Conexión MQTT:** Establece la conexión con el *broker* MQTT y se suscribe a los tópicos de tienda/pedidos y tienda/respuesta para poder enviar y recibir mensajes.
 - **Inicialización de la Cámara:** Configura todos los parámetros de la cámara (resolución, formato, pines, etc.) para que pueda tomar imágenes.
 - **Creación de la Tarea del Lector QR (xTaskCreatePinnedToCore):** Arranca la función `QRCodeReader` como una tarea separada, permitiendo que la lectura de QR se ejecute en segundo plano sin bloquear el programa principal.
- **loop(): El Ciclo Continuo del Sistema**
 - Se ejecuta **repetidamente sin parar** después de que `setup()` termina.

- **client.loop():** Es vital para la conexión MQTT. Esta línea **mantiene la conexión con el *broker* MQTT** y revisa si hay mensajes nuevos que procesar (activando el callback()).
- **Procesamiento de Pedidos y QR:**
 - Verifica si no está esperando un QR y si hay pedidos en la cola. Si es así, toma el siguiente pedido y lo inicia (procesarPedido).
 - Si la tarea del lector QR ha encontrado un QR (qrLeido = true), se encarga de enviar el pedido completo con el QR.
- **delay(10):** Una pequeña pausa para que el ESP32 no consuma todos sus recursos y permita que otras tareas (como la del lector QR) también tengan tiempo de ejecutarse.