Proyecto Final – Recolección y Reciclaje Inteligente

Documentación del Proyecto

Alumnos: Ferrando, Iannone, Manfredo, Montenegro y Siciliano

6° Informática B - Grupo 8

Profesores: Barral, Martini, Pedaci, Santisi, Vivani y Werner

*Instituto Leonardo Murialdo*

**Índice**

Contenido

[Proyecto Final: Reciclaje Inteligente 3](#_Toc211850908)

[**Objetivos** 4](#_Toc211850909)

[Generales 4](#_Toc211850910)

[Específicos 5](#_Toc211850911)

[**Alcance** 5](#_Toc211850912)

[Alcance actual 5](#_Toc211850913)

[Fuera del alcance (hoy) 6](#_Toc211850914)

[Criterios de cierre 6](#_Toc211850915)

[**Marco Teórico** 7](#_Toc211850916)

[Capa física (sensado y actuación) 7](#_Toc211850917)

[Alimentación y compatibilidad electromagnética 7](#_Toc211850918)

[Nodo IoT (ESP32) y control en firmware 8](#_Toc211850919)

[Backend (Flask + SQLite) y contrato de datos 8](#_Toc211850920)

[Dashboard (PWA) y experiencia operativa 9](#_Toc211850921)

[Cambios recientes consolidados 9](#_Toc211850922)

[Flujo funcional (de señales a acción) 9](#_Toc211850923)

[Temporización y robustez 10](#_Toc211850924)

[**Componentes** 10](#_Toc211850925)

[Principios de diseño de la tabla de costos 10](#_Toc211850926)

[Bloques y elecciones clave 11](#_Toc211850927)

[**Diagrama** 13](#_Toc211850928)

[Pinout (ESP32 WROOM / NodeMCU ESP32S) 13](#_Toc211850929)

[Alimentación y dominios 14](#_Toc211850930)

[Paso a paso de conexión 14](#_Toc211850931)

[Chequeo final 18](#_Toc211850932)

[**Funcionamiento del tacho** 19](#_Toc211850933)

[**Tabla de Costos** 20](#_Toc211850934)

[**Carta GANTT** 20](#_Toc211850935)

[**Bitácora** 21](#_Toc211850936)

[**Desarrollo Técnico** 23](#_Toc211850937)

[**Conclusión** 40](#_Toc211850938)

[¿Qué se logró? 40](#_Toc211850939)

[¿Qué no pudo lograrse? 41](#_Toc211850940)

[¿Por qué no pudo lograrse? 41](#_Toc211850941)

[¿Cómo podría mejorarse? 42](#_Toc211850942)

Proyecto Final: Reciclaje Inteligente

El proyecto “Reciclaje y Recolección Inteligente” evoluciona a una arquitectura ciber-física IoT centrada en tres fracciones de residuo (Metal, Orgánico y Resto) y en un eje lineal con correa accionado por un motor paso a paso unipolar de 6 hilos (ULN2003 a 9 V). El objetivo es sustituir estimaciones visuales y recorridos reactivos por un flujo medible y explicable que detecta el ingreso de un objeto, lo clasifica con reglas físicas claras, posiciona la base móvil bajo el contenedor correcto y confirma el depósito con pesaje en gramos para calcular el porcentaje de llenado de cada tacho.

En el borde, un ESP32 coordina una máquina de estados. La detección de presencia se realiza con un sensor IR que abre una breve ventana de muestreo. La clasificación es determinista y priorizada: el sensor inductivo NPN, alimentado a 12 V y adaptado a lógica de 3,3 V, identifica metales de forma binaria y robusta; el sensor SHT31 detecta orgánicos a partir de variaciones de temperatura y humedad respecto de un baseline exponencial que filtra la deriva ambiental; si no se cumplen estas condiciones, el material se etiqueta como Resto.

La actuación mecánica se basa en una base que se desplaza horizontalmente por correa. El motor paso a paso se gobierna con AccelStepper en modo no bloqueante, con rampas de aceleración y velocidad máximas parametrizables, expresadas en mm/s y mm/s². El sistema ejecuta homing al encender contra un final de carrera para fijar el cero lógico; a partir de ese origen, las posiciones de los tres tachos (espaciados y ajustables en milímetros) se almacenan de modo persistente y pueden recalibrarse sin recompilar. La compuerta de descarga abre mediante un microservo gobernado por un PCA9685 a 50 Hz, lo que asegura un PWM estable y desacoplado de la carga de CPU del ESP32.

La confirmación del depósito se realiza por peso: cada tacho descansa sobre una celda de carga de 5 kg conectada a su propio HX711. Tras la caída, el firmware mide Δg, actualiza gramos acumulados y traduce ese valor a porcentaje de ocupación respecto de la capacidad nominal de cada contenedor. Un buzzer de 5 V y dos tiras NeoPixel proporcionan señalización local; la línea DIN de los LEDs se adapta 3,3 → 5 V mediante 74HCT14 con resistencia serie y el ramal de 5 V incorpora 1000 µF de desacople para absorber picos. La alimentación separa lógica a 3,3 V de potencia a 5/9/12 V, con masa común y retornos cortos para mitigar EMI.

Sobre esta base física, una API web en Flask con SQLite persiste la configuración y los eventos. El contrato incluye GET/POST /api/config para pasos/mm, Vmax, Amax y posiciones de los tres tachos; POST /api/deposits para registrar cada depósito con sello temporal, tacho, clase y peso; GET /api/bins para exponer estado agregado por contenedor; y GET/POST /api/axis/state para la telemetría del eje (estado, homing y posición). Los POST desde el firmware implementan verificación de código HTTP y reintentos con backoff, manteniendo una pequeña cola en RAM si la red no responde, de modo que control y seguridad no dependan de la conectividad.

La interfaz es un dashboard PWA responsive que se instala en móvil o escritorio. Presenta tarjetas por tacho con barras de progreso, gramos y alertas a partir de un umbral configurable; KPIs diarios (peso total, cantidad de eventos, material más frecuente); tabla de eventos recientes; y un gráfico histórico de gramos por día con selector de rango. Un panel de diagnóstico muestra el estado del eje en tiempo real; cuando la API expone WebSocket, la UI actualiza posición y estado sin recargas. El diseño visual adopta un tema claro/oscuro, tipografía legible y componentes con jerarquía de información para minimizar tiempo-a-insight.

El proyecto privilegia la explicabilidad y la operabilidad: cada decisión deriva de mediciones físicas y umbrales documentados dentro de ventanas temporales conocidas; el eje se referencia por homing reproducible; los pesos son trazables y exportables; la red se trata como mejor-esfuerzo con degradación controlada. La arquitectura se mantiene modular: la base mecánica y de sensado es sencilla y robusta, el firmware es parametrizable, la API es estable y migrable a PostgreSQL, y la PWA puede crecer con notificaciones, analítica o integraciones sin cambiar los cimientos. El resultado es un sistema que reduce desbordes y recorridos innecesarios, planifica vaciados con evidencia y facilita auditoría y mantenimiento en escenarios reales.

## **O****bjetivos**

Generales

Digitalizar y optimizar la gestión de residuos cerrando el ciclo entre detección y acción en el borde (clasificación determinista y operación mecatrónica casi en tiempo real sobre un eje lineal con correa y motor paso a paso), observabilidad centralizada (persistencia y consulta de eventos/estados en una API Flask con almacenamiento transaccional) y toma de decisiones operativas basada en KPIs (cuándo vaciar, qué ruta priorizar, dónde se satura antes). El sistema no solo alerta: explica por qué alerta con evidencia verificable (registro del evento, masa depositada, % de ocupación calculado contra capacidad nominal y serie temporal para tendencia), asegurando trazabilidad completa desde la medición física hasta la visualización en el dashboard PWA.

### Específicos

Medir el peso por tacho con celdas de 5 kg y HX711 (resolución a nivel de gramos), derivar el % de ocupación con criterio operativo (capacidad nominal por tacho y/o coeficientes de calibración) y aplicar filtrado adecuado para estabilidad sin introducir latencia perceptible al operador: EMA/mediana deslizante en ventanas cortas que eliminan jitter sin comprometer respuesta. Con montaje correcto y calibración de dos puntos por canal se apunta a un error típico ≤ ±3 %. Implementar alerta por umbral configurable (por defecto 80 %) porque a partir de ese nivel cae la usabilidad y sube el riesgo de desborde; la UI debe reflejar el cambio de estado en < 2 s, incorporando histéresis para evitar parpadeos cuando la señal oscila cerca del umbral.

Proveer una visualización integral en un dashboard PWA instalable y responsive que muestre KPIs de uso, tarjetas por tacho con barras y badges semánticos, y un gráfico diario/histórico; habilitar exportación CSV y un panel de Ajustes que persiste parámetros vía API (pasos/mm, Vmax, Amax, posiciones de los 3 tachos, umbrales). Mantener un modo demo robusto para operar aun sin backend (datos sintéticos y mensajes explícitos de estado), con conmutación transparente a modo online sin recargar, preservando estado local y minimizando tiempo fuera de servicio.

## **Alcance**

### Alcance actual

El alcance actual fija el hardware y la arquitectura de control: ESP32 como nodo IoT; sensor IR para disparo de ventana de muestreo; sensor inductivo NPN (12 V, salida adaptada a 3,3 V) para detección binaria de metales; SHT31 para orgánicos vía variaciones de ΔT/ΔHR sobre un baseline exponencial; tres celdas de carga de 5 kg con HX711 (una por tacho) para confirmar depósito y computar % de ocupación; PCA9685 para el servo de compuerta (PWM estable a 50 Hz); NeoPixel con SN74HCT14N como level shifter (3,3→5 V) para señalización; y eje lineal con motor paso a paso unipolar + ULN2003 con homing a X\_min. Se entrega firmware con FSM (ventanas temporales, prioridad Metal > Orgánico > Resto, cinemática no bloqueante con AccelStepper, publicación HTTP/JSON, reintentos con backoff y watchdog), backend Flask + SQLite con entidades events, bins (peso/capacidad/%), config (umbrales, pasos/mm, Vmax, Amax, posiciones) y telemetría del eje; además de OpenAPI básico. Sobre esa API, un dashboard PWA instalable y responsive muestra KPIs, tarjetas por tacho con barras/alertas, gráfico diario/histórico, exportación CSV, Ajustes persistentes y modo demo. La documentación incluye armado/cableado, calibración HX711 a dos puntos, criterios de seguridad eléctrica (dominios 3,3 V/5–12 V, GND común, desacoples) y plan de pruebas funcionales. Se asume instalación interior/semicubierta, fuente 5 V/≥3 A para actuadores/LEDs, lógica a 3,3 V, y montaje mecánico que garantice linealidad y baja fricción en celdas.

### Fuera del alcance (hoy)

Queda fuera del alcance en esta fase: compactación mecánica (requiere mecánica de potencia), balanzas certificadas legales (metrología reglamentaria), y clasificación por IA/visión (se mantiene en roadmap; el MVP privilegia sensores físicos por costo/explicabilidad/robustez). También se excluyen integraciones complejas (ERP municipal, SSO corporativo, mensajería masiva); la API actual es la base para evolucionar hacia esos conectores sin reescribir firmware ni UI.

### Criterios de cierre

Los criterios de cierre comprueban entregables y desempeño: firmware estable con FSM, parámetros editables y publicación verificada; API Flask operativa con events/bins/config y DB auto inicializada; dashboard PWA con KPIs, tarjetas, gráfico, Ajustes y modo demo validado; calibración HX711 reproducible (error típico ≤ ±3 % con montaje correcto); pruebas de umbral (80 % con histéresis) y degradación controlada ante fallas de red; y documentación técnica completa (armado, seguridad y pruebas de aceptación) que permita replicar, operar y mantener el sistema.

## **Marco Teórico**

### Capa física (sensado y actuación)

Cada contenedor (Metal, Orgánico y Resto) se apoya sobre una celda de carga de 5 kg leída por un HX711 dedicado. Este convertidor provee ganancia programable y entrada diferencial apta para microvoltios típicos de galgas; no obstante, la precisión final depende más del montaje que del ADC. Por eso se aplica una calibración a dos puntos por tacho (offset con tacho vacío y escala con masa patrón) y se exige mecánica rígida: sin flexión lateral, sin roces, con apoyo plano y guiado que limite fuerzas parásitas.

En cableado se recomiendan tramos cortos, masa en estrella, evitar bucles y, si las corridas superan decenas de centímetros, blindaje básico o trenzado. La detección de presencia IR actúa como compuerta lógica (gate): confirma que hay un objeto en boca y abre una ventana breve de clasificación, eliminando “latigazos” y evitando decisiones fuera de contexto. La detección de metal se realiza con un sensor inductivo NPN alimentado a 12 V; su salida se adapta a lógica 3,3 V (divisor, NPN u opto) y se trata de forma binaria y determinista, dándole prioridad máxima para cerrar rápido la decisión cuando aplica. La detección de orgánicos se apoya en el SHT31: el firmware calcula ΔT y ΔHR respecto de un baseline exponencial, lo que desacopla la deriva ambiental lenta del microclima que producen residuos orgánicos. Si no se cumple metal ni orgánico, el material se clasifica como Resto.

En la actuación, la base móvil se desplaza horizontalmente con motor paso a paso unipolar (6 hilos) + ULN2003 y correa con polea de retorno (proveniente de mecánica de impresora) para posicionar el conducto sobre el tacho objetivo; la compuerta abre con un microservo gobernado por PCA9685 (PWM estable a 50 Hz, calibración por microsegundos). Para feedback local, un buzzer de 5 V emite confirmaciones y una tira NeoPixel (WS2812) en 5 V aporta señalización visual; la línea DIN se nivela 3,3→5 V con SN74HCT14N (dos puertas en serie para no invertir), se coloca R serie 300–500 Ω y 1000 µF de desacople en 5 V de la tira.

### Alimentación y compatibilidad electromagnética

Se separan dominios: lógica 3,3 V (ESP32, I²C) y potencia 5–12 V (NeoPixel, buzzer, servo, motor a 9–12 V), con GND común y retornos cortos. Se distribuyen 100 nF de bypass por cada CI y 1000 µF en el ramal de 5 V de LEDs; las corrientes del motor/servos se rutean lejos de señales sensibles (HX711/I²C). En buses I²C con varias placas, se evita la suma de pull-ups: se deja un único par efectivo para mantener flancos limpios y timing estable.

### Nodo IoT (ESP32) y control en firmware

El ESP32 (WROOM) aporta Wi-Fi y doble core, facilitando separar tiempo real de lógica/red. El control sigue una FSM explícita con temporalidades documentadas: IDLE (espera) → ENTRY\_SAMPLE (≈ 600 ms de ráfagas sobre inductivo y SHT31 con antirrebote por densidad de muestras) → AIM (posicionamiento de la base) → RELEASE (apertura breve de compuerta) → WEIGHING (asentamiento corto y lectura Δg) → RESET (retorno a reposo y cooldown). El eje lineal se gobierna con AccelStepper en modo no bloqueante, con rampas de velocidad y aceleración expresadas en mm/s y mm/s²; al encender se ejecuta homing contra un final de carrera X\_min para fijar el cero lógico y, desde allí, se alcanzan las tres posiciones de tacho (en milímetros, editables). Las lecturas son no bloqueantes, los umbrales/tiempos se parametrizan vía API y persisten en backend, y la publicación HTTP/JSON implementa verificación de estado, reintentos con backoff y watchdog, de modo que la red sea “best effort” sin impactar control ni seguridad.

### Backend (Flask + SQLite) y contrato de datos

SQLite aporta persistencia real sin infraestructura. La API modela un conjunto mínimo y suficiente:

* config: parámetros operativos (pasos/mm, Vmax, Amax, posiciones de los 3 tachos, umbral de alerta, etc.) editables vía GET/POST /api/config;
* bins: estado agregado por tacho (gramos, capacidad y % derivado), expuesto en GET /api/bins;
* events: registro de depósitos con sello temporal, tacho, clase y Δg publicados desde el firmware en POST /api/deposits;
* axis/state: telemetría del eje (estado, homed y posición) en GET/POST /api/axis/state.

El contrato OpenAPI básico documenta rutas y estructuras, la auto inicialización de la DB simplifica despliegue y la migración a PostgreSQL queda habilitada sin cambiar firmware ni UI.

### Dashboard (PWA) y experiencia operativa

La interfaz prioriza tiempo-a-insight bajo: tarjetas por tacho con barras y badges (OK/Warning/Alert) actualizadas en segundos, KPIs diarios (peso acumulado, cantidad de eventos, material más frecuente) y gráfico de evolución con selector de rango. Un panel de Ajustes permite editar URL de la API y parámetros (umbrales, cinemática y posiciones) y persiste vía /api/config. Soporta exportación CSV, modo demo cuando la API no responde (datos sintéticos y mensajes claros), y comportamiento PWA (manifest + service worker) para instalación en móvil/escritorio. Estilo responsive con tema claro/oscuro, paleta verde/negro y tipografía legible; animaciones suaves refuerzan estados sin distraer.

### Cambios recientes consolidados

Se eliminó el canal “Plástico/Vidrio” y todo lo asociado a LDR (el sensor de peso correspondiente al tacho, el actuador ADS1115); la prioridad de clases es Metal > Orgánico > Resto; el IR quedó solo como trigger de ventana; la apertura se realiza con compuerta servo-controlada (no hay RCWL por tacho); la base móvil ahora se posiciona con motor paso a paso + ULN2003; se confirmó SN74HCT14N (DIP-14) como level shifter para NeoPixel; y todos los parámetros (umbrales y cinemática) son ajustables vía API sin re flasheo.

### Flujo funcional (de señales a acción)

El IR detecta presencia y abre una ventana de clasificación (~600 ms) durante la cual el firmware toma ráfagas de muestras con antirrebote temporal. En la cascada determinista, si el inductivo indica metal (tras adaptación a 3,3 V), se cierra de inmediato con prioridad máxima; si no, se evalúan ΔT/ΔHR contra el EMA del SHT31 y, de superar umbrales, se cataloga Orgánico; si ninguna regla aplica, se etiqueta como Resto. Decidido el material, la FSM ordena al eje lineal moverse al setpoint (mm) del tacho correspondiente con AccelStepper; alcanzada la posición, la compuerta abre un tiempo breve y cierra. Tras un asentamiento corto, el HX711 del tacho afectado mide y se calcula Δg, se actualizan gramos y % de ocupación (capacidad nominal 5 kg por tacho) y se publica el evento en /api/deposits. Finalmente, el sistema retorna a reposo y aplica un cooldown que amortigua rebotes mecánicos o térmicos antes del siguiente ciclo.

### Temporización y robustez

La ventana de clasificación equilibra respuesta humana fluida con suficiente densidad de evidencia; la UI aplica histéresis para evitar parpadeos cerca del 80 %; la publicación a la API incorpora reintentos y watchdog para que caídas de red no detengan lazo de control; y todos los umbrales/tiempos son editables por API, habilitando afinación sin recompilar. El resultado es una arquitectura modular y explicable que convierte señales físicas en acciones mecánicas y decisiones operativas, con trazabilidad de extremo a extremo y configuración centralizada, alineada con todos los cambios recientes del proyecto.

Una arquitectura modular, explicable y operativa que convierte señales físicas en acciones mecánicas y decisiones operativas, con trazabilidad de extremo a extremo y configuración centralizada, incorporando todos los cambios recientes del proyecto.

## **Componentes**

La tabla de costos se diseñó con un criterio doble: robustez en lo esencial (sensado, actuación, alimentación) y bajo costo en la periferia. Esto permite que las decisiones que más afectan confiabilidad, precisión y mantenibilidad queden blindadas, mientras que el resto se mantiene simple y económico. A continuación, se detalla el porqué de cada elección, las implicancias técnicas, los pasivos necesarios (no siempre cotizados) y las equivalencias posibles, incorporando los cambios recientes del proyecto.

### Principios de diseño de la tabla de costos

* **Primero lo crítico**: sensores de clase, medición de peso (amplificador HX711 + celdas), mecatrónica (servo selector + tapas), alimentación y EMC/ruido.
* **Simplicidad explicable**: evitar componentes que agreguen complejidad sin mejorar la trazabilidad (p. ej., IA/visión en esta etapa).
* **Prototipado y servicio**: preferencia por encapsulados DIP cuando no penalizan performance para facilitar protoboard y mantenimiento.
* **Parametrización por software**: umbrales/tiempos se afinan desde la API.

### Bloques y elecciones clave

* **Level shifter / Acondicionamiento digital**
  + **Elegido:** **SN74HCT14N (DIP-14)** —familia HCT (umbrales TTL a 5 V).
  + **Motivo técnico:** con entradas Schmitt limpia flancos y rechaza ruido; al trabajar a 5 V interpreta el 3,3 V del ESP32 como “1 lógico” sin margen dudoso.
  + **Uso en NeoPixel (WS2812, 5 V):** dos puertas en serie (no invertir la señal), R serie 300–500 Ω en DIN y 1000 µF cerca de la tira.
  + **Cambio reciente incorporado:** se consolidó SN74HCT14N como estándar (antes “74HCT14” genérico).
* **Servomotores**
  + **Tipo:** SG92R 9 g (≈25 kg·cm 4,8 V; referencia típica del modelo).
  + **Razón:** par suficiente para tapas livianas y selector liviano; bajo consumo y disponibilidad alta.
  + **Control**: PCA9685 (PWM estable, 16 canales), alejando el timing del load de CPU del ESP32; posiciones en µs para OPEN/CLOSE/REST.
  + **Notas:** fuentes/tomas de 5–6 V para servos independientes del 3,3 V del ESP32; GND común.
* **Sensores de presencia y clasificación**
  + **IR de obstáculos**: trigger inequívoco para abrir ventana de clasificación.
  + **Inductivo NPN (LJ12A3-4-Z/BX)**: metal con prioridad máxima. Lectura active-low; adaptar nivel si OUT > 3,3 V.
  + **SHT31**: orgánico por ΔT/ΔHR vs baseline exponencial (α ≈ 0,005). Umbrales ajustables por API
* **Medición de peso**
  + **Celdas de carga 5 kg + HX711** (por tacho).
  + **Razón:** ganancia programable y lectura diferencial (µV).
  + **Clave de precisión:** apoyo rígido, sin flexión lateral ni rozamientos; cables cortos; masa en estrella; calibración dos puntos por tacho.
* **Alimentación y EMC**
  + **Dominios separados:** 3,3 V (lógica/ESP32/I²C) vs 5 V (servos, NeoPixel, buzzer). GND común.
  + **Desacople mínimo recomendado:** 100 nF por IC; 1000 µF en la línea de 5 V de NeoPixel; inyección adecuada de potencia si la tira crece.
  + **Routing**: separar retornos de potencia de líneas sensibles (HX711, LDR).
  + **Pull-ups I²C:** evitar múltiples pull-ups en paralelo; dejar un par efectivo.

**Pasivos y mecánica.**

* **Resistencias:** 300–500 Ω (DIN NeoPixel), 10 kΩ, 1–4,7 kΩ (bases NPN, si se usan).
* **Capacitores:** 100 nF por IC; 1000 µF/6,3–10 V en 5 V de LED/servos.
* **Protecciones/otros:** diodo de rueda libre si se usa carga inductiva; fusible/limitador de 5 V (opcional según instalación).
* **Mecánica:** bastidor rígido; guías del selector con topes definidos; montaje de celdas sin cargas laterales; tornillería adecuada.
* **Cableado:** secciones acordes a corriente de servos/LED; crimpados/headers firmes; longitudes cortas en señales sensibles.

**Variantes y equivalencias.**

* **74HCT14** ⇄ **SN74HCT14N** (DIP-14) — equivalentes funcionales.
* **PCA9685**: múltiples fabricantes; mantener 50 Hz para servos.
* **HX711**: módulos de distintos proveedores; verificar ganancia y pinout DT/SCK.
* **Servomotor**: si se reemplaza por otro microservo, validar par y consumo; recalibrar µs.

**Consideraciones de compra y mantenimiento.**

* **Stock crítico de repuesto**: 1–2 servos extra, 1 HX711, 1 SN74HCT14N y tira NeoPixel corta.
* **Trazabilidad:** etiquetar celdas por tacho y documentar fecha de calibración.

**Impacto de las últimas decisiones en la tabla de costos.**

* SN74HCT14N confirmado como level shifter estándar (DIP-14) → prototipado fácil, limpieza de flancos, dos puertas para no invertir.
* Parámetros por API (/api/config) → la tabla de costos no depende de “ajustes duros”; los umbrales/tiempos se calibran en operación.

## **Diagrama**

### Pinout (ESP32 WROOM / NodeMCU ESP32S)

* **I²C (3.3 V lógica)**
  + SDA → GPIO 21
  + SCL → GPIO 22
  + Dispositivos en el bus: SHT31 (0x44) y PCA9685 (0x40)
* **Sensores digitales (entradas)**
  + IR obstáculo (hay objeto) → GPIO 26 (entrada digital)
  + Inductivo NPN LJ12A3-4-Z/BX (detectar metal) → GPIO 27 (entrada digital, ver adaptación a 3.3 V abajo)
  + Endstop X\_min (NC, activo-bajo) → GPIO 16 (entrada con INPUT\_PULLUP)
* **Señalización / actuadores lógicos**
  + NeoPixel DIN (vía 74HCT14 + R serie 330–470 Ω) → GPIO 23 (salida)  
    Buzzer (a través de NPN + R base) → GPIO 2 (salida)
* **Balanzas / llenado (3 tachos)**
  + HX711 SCK (común a los 3 módulos) → GPIO 25 (salida)
  + HX711 DOUT Tacho 1 (Metal) → GPIO 34 (solo entrada)
  + HX711 DOUT Tacho 2 (Orgánico) → GPIO 35 (solo entrada)
  + HX711 DOUT Tacho 3 (Resto) → GPIO 36 (solo entrada)
* **Eje lineal (ULN2003 + paso a paso 6 hilos)**
  + ULN2003 IN1 → GPIO 18 (salida)
  + ULN2003 IN2 → GPIO 19 (salida)
  + ULN2003 IN3 → GPIO 5 (salida)
  + ULN2003 IN4 → GPIO 17 (salida)

*(En firmware se usa AccelStepper FULL4WIRE con orden IN1, IN3, IN2, IN4.)*

* **Servo “gate” (abre compuerta)**
  + PCA9685 canal CH0 → microservo de compuerta (PWM 50 Hz)

### Alimentación y dominios

* 3.3 V (lógica, desde el ESP32): ESP32, SHT31, PCA9685 (VCC, no V+), HX711 (recomendado a 3.3 V para compatibilidad de niveles).
* 5 V (potencia liviana): NeoPixel, V+ de PCA9685 (servo gate).
* 9–12 V (motor paso a paso): alimentación del motor al ULN2003 (recomendado 9 V según tu motor).
* 12 V (sensor inductivo NPN): el LJ12A3 suele requerir 6–36 V. 12 V es típico.

**Masa común entre todas las fuentes (3.3 V, 5 V, 9/12 V) y retorno corto.**

**Desacople: 100 nF por IC; 1000 µF entre 5 V y GND cerca de la tira NeoPixel; 100–220 µF cerca del ULN2003.**

**Protección: fusible/PTC en 5 V (y en 9/12 V si es posible).**

### Paso a paso de conexión

**1) Preparar fuentes y masas**

1. Colocá tus fuentes: 3.3 V (ESP32), 5 V ≥ 3 A (NeoPixel/servo), 9 V (motor), 12 V (sensor inductivo).
2. Uní todas las GND en un punto común (estrella) para evitar lazos.
3. Agregar 1000 µF/10 V entre 5 V y GND junto a la tira NeoPixel; 100 nF cerca de cada PCA9685, SHT31 y cada HX711.
4. Verificar polaridades y continuidad.

**2) Bus I²C (SHT31 + PCA9685)**

1. Conectar SDA (GPIO21) y SCL (GPIO22) del ESP32 a SDA/SCL de SHT31 y PCA9685 en paralelo.
2. VCC (3.3 V) a VCC de ambos; GND común.
3. En el PCA9685:
   1. VCC = 3.3 V (lógica I²C).
   2. V+ (servo power) = 5 V (no lo mezcles con VCC).
   3. GND de V+ y GND de lógica deben estar en común.
4. Direcciones por defecto: SHT31 0x44, PCA9685 0x40

**3) Servo “gate” (PCA9685 CH0)**

1. Conectar el microservo (señal al pin CH0 del PCA9685, rojo a 5 V+, negro a GND).
2. En el firmware ya están mapeados los pulsos (≈500–2400 µs mapeados a ticks del PCA).

**4) NeoPixel con 74HCT14 (3.3→5 V)**

* ESP32 GPIO23 → entrada 1 del 74HCT14N → salida 1 → entrada 2 → salida 2 → R serie 330–470 Ω → DIN de la tira.  
  *(Uso de 2 puertas en serie para no invertir la señal.)*
* Tira NeoPixel: 5 V y GND robustos; 1000 µF cerca; no alimentarla desde el ESP32.
* GND de la tira común con el ESP32.

**5) Buzzer 5 V con NPN**

1. ESP32 GPIO2 → R base 1 kΩ → base de 2N2222/SS8050.
2. Emisor a GND; colector al terminal negativo del buzzer; terminal positivo a 5 V.
3. Si el buzzer es inductivo (no piezo activo), agregar diodo 1N4148/1N5819 en antiparalelo (cátodo a +5 V).

**6) Sensores de presencia/clasificación**

**6.1 IR de obstáculo (trigger de ventana de muestreo)**

1. VCC: preferí 3.3 V (si el módulo lo soporta) para que la salida sea 3.3 V segura.  
   Si usás 5 V, bajá la salida a 3.3 V con un divisor 10 k/20 k o transistor.
2. GND común.
3. OUT → GPIO26.

**6.2 Inductivo NPN LJ12A3-4-Z/BX (detectar metal)**

1. Marrón a +12 V, Azul a GND.
2. Negro (OUT): es colector abierto. Dos opciones seguras:
   1. Adaptación simple: OUT al GPIO27 con pull-up de 10 kΩ a 3.3 V (y una R serie 1 kΩ al GPIO). *GND en común con 12 V y 3.3 V.*
   2. Aislamiento (recomendado): OUT → optoacoplador → ESP32 (3.3 V).
3. Lógica en firmware: HIGH = metal (active-high tras el pull-up a 3.3 V).

**6.3 Endstop X\_min (NC, activo-bajo)**

1. Común del switch a GND; contacto NC al GPIO16.
2. Configurar el pin con INPUT\_PULLUP (ya está en firmware).
3. Cuando “toca”, el pin lee LOW

**7) Balanzas (3× HX711 + celdas de carga)**

1. Usar 1 HX711 por tacho. Aquí se asume 3 módulos.
2. Alimentación HX711: 3.3 V + GND (para que DOUT/SCK estén en 3.3 V).
3. Señales:
   1. SCK (común) → GPIO25.
   2. DOUT Tacho 1 (Metal) → GPIO34.
   3. DOUT Tacho 2 (Orgánico) → GPIO35.
   4. DOUT Tacho 3 (Resto) → GPIO36.
4. Celdas: cable corto, fijación rígida, sin cargas laterales. A+ / A- a la galga; E+ / E- al puente.
5. Calibración: offset con tacho vacío; escala con masa patrón; guardar en NVS o en firmware.

**8) Motor paso a paso 6 hilos + ULN2003 (eje horizontal)**

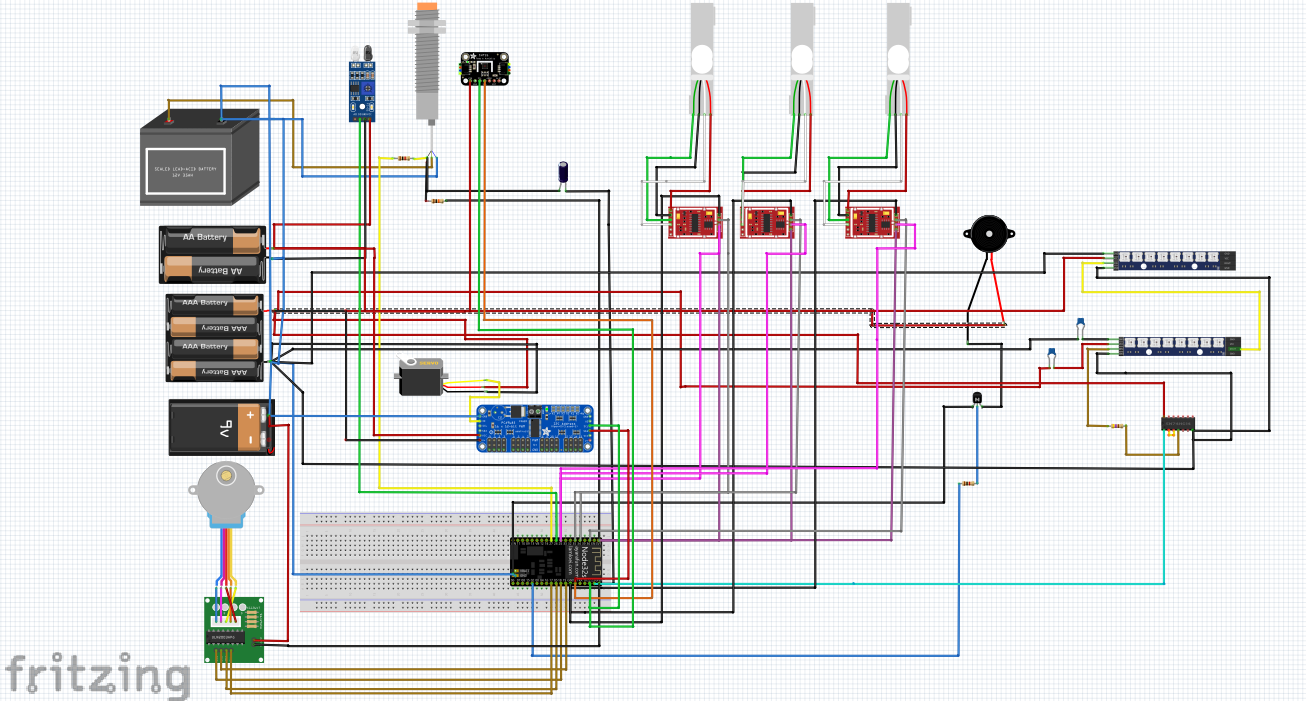
1. Motor 6 hilos unipolar (dos comunes al +V, 4 bobinas a colectores). En la placa ULN2003:
   1. Conecta las 4 bobinas del motor a las salidas del ULN2003 (OUT1..OUT4).
   2. Uní los dos comunes del motor al +9 V (o el valor nominal de tu motor).
2. Entradas ULN2003 desde el ESP32:
   1. IN1 ← GPIO18
   2. IN2 ← GPIO19
   3. IN3 ← GPIO5
   4. IN4 ← GPIO17
3. GND del ULN2003 común con el ESP32 y con la fuente del motor.
4. Alimentación del motor: +9 V al pin VM del ULN2003 / común del motor.
5. Condensador de 100–220 µF cerca de la bornera del ULN2003.
6. En firmware se usa AccelStepper (FULL4WIRE, IN1, IN3, IN2, IN4) para el orden correcto y movimiento no bloqueante.

**9) Distribución física / posiciones**

1. Distancia entre centros de tachos: 12 cm (120 mm).
2. Anchura de cada tacho: ~10 cm.
3. pos\_mm\_bins (firmware / API): {0, 120, 240} mm, medidos desde el cero hecho por homing (X\_min).
4. En homing: el eje baja a X\_min, libera (bounce), “kiss” lento, fija posición 0 y se posiciona al centro del Tacho 1.

### Chequeo final

* **Fuentes separadas: 3.3 V (lógica), 5 V (NeoPixel/servo), 9 V motor, 12 V inductivo. GND común.**
* **Desacoples: 100 nF por IC; 1000 µF en 5 V de la tira; 100–220 µF en ULN2003.**
* **NeoPixel con 74HCT14N (dos puertas) + R 330–470 Ω en DIN.**
* **Inductivo con pull-up a 3.3 V + R serie 1 kΩ u opto (recomendado).**
* **HX711 a 3.3 V; SCK común (GPIO25); DOUT en GPIO34/35/36.**
* **I²C correcto: SDA21/SCL22 → SHT31 (0x44) y PCA9685 (0x40).**
* **ULN2003 IN1..IN4 en 18,19,5,17; motor a +9 V; GND común.**
* **Endstop X\_min NC a GPIO16 con INPUT\_PULLUP.**
* **Servo gate en PCA9685 CH0; V+ 5 V para potencia de servo.**
* **Verificar pos\_mm\_bins = [0,120,240] en dashboard/API/firmware.**

**[](https://d.docs.live.net/d0d228de41100acb/Escritorio/Proyecto%20Final%20-%20Recoleccion%20y%20Reciclaje/Diagrama%20-%20fritzing/proyecto%20-%20otra%20version%20(1).fzz)**

**Aclaración: No se pudo encontrar un componente similar al final de carrera para añadir al diagrama, pero dejamos como es el pinout para demostrar cómo es su conexión**

**Endstop (o Final de Carrera) X\_min (NC, activo-bajo)**

1. **Común del switch a GND; contacto NC al GPIO16.**
2. **Configura el pin con INPUT\_PULLUP (ya está en firmware).**
3. **Cuando “toca”, el pin lee LOW.**

## **Funcionamiento del tacho**

**Primero:**

* Colocar la basura que quiera tirarse en el tacho.

**Segundo:**

* Un sensor infrarrojo detectará que la basura que este contenida en el tacho, activando el resto de sensores.

**Tercero:**

* Mediante sensores de temperatura/humedad y de metales (inductivo), el tacho detectará automáticamente que tipo de basura es.

**Cuarto:**

* Detectada la basura, el motor paso a paso (con la ayuda del driver ULN2003) se activa hasta que el tacho correspondiente quede debajo del contenedor.

**Quinto:**

* La puerta del contenedor superior se abrirá a través de un servomotor (apoyado por el PCA9685), dejando caer la basura a su tacho específico, y se volverá a cerrar posteriormente.

**Sexto:**

* Un sensor de peso y un amplificador (HX711), detectarán el peso correspondiente de la cantidad total de basura dispuesta en cada tacho y el nivel de llenado de estos mismos.
* Así mismo, si un tacho supera un límite de llenado asignado, se activará un buzzer y la cantidad de leds correspondientes

**Séptimo:**

* Tal información captada en la maqueta, se visualizará en el dashboard, la cual se conecta al código del ESP32 gracias a la API.

## **Tabla de Costos**

## **Carta GANTT**

* [Planner - Smart Cities - Reciclaje y Recolección Inteligente](https://teams.microsoft.com/l/entity/com.microsoft.teamspace.tab.planner/planner.v1.95d94ae7-1356-499a-a5bd-aacd9f060443_p_aTHcOEjtNUClzzs-qVqSUmUAByHe?tenantId=bf9504f3-4385-4e7e-b7ec-5997e0e6b0a5&webUrl=https%3A%2F%2Ftasks.teams.microsoft.com%2Fteamsui%2FpersonalApp%2Falltasklists&context=%7B%22subEntityId%22%3A%22%2Fv1%2Fplan%2FaTHcOEjtNUClzzs-qVqSUmUAByHe%22%2C%22channelId%22%3A%2219%3AsVyLkJ47Jx3qQNTDjhzqiGfBYQGz2j-OkzUPJgocy5w1%40thread.tacv2%22%7D)

## **Bitácora**

(decisiones, iteraciones y aprendizajes)

**Planteo (semana 1).**

Se definió el problema operativo: *“necesitamos saber cuándo y dónde vaciar, sin mandar a alguien a mirar todos los tachos”*. Se eligió peso como métrica principal por ser objetiva. Se descartó empezar por visión/IA: costo y complejidad no aportaban al MVP.

**Selección tecnológica (semana 2).**

* Se cerró en ESP32 por Wi-Fi integrado y ecosistema. Para sensado: HX711 (fiable, económico), IR (trigger), Inductivo (metal con alta precisión), SHT31 (orgánico por ΔT/ΔHR), LDR+LED (plástico/vidrio por umbral). RCWL por canal se adoptó para abrir tapa al arribo y desacoplar clasificación de operación (clave para robustez).

**Arquitectura y FSM (semana 3).**

* Se diseñó la FSM con estados y tiempos explícitos (clasificación, ruta, apertura, cooldown). Se estableció prioridad de decisiones para evitar conflictos. Se documentó el rol del IR como *trigger* (antes se había intentado usar proximidad para todo y generaba ambigüedad).

**Electrónica (semana 4).**

* Se separaron dominios 3,3 V/5 V y se definió GND común. Para NeoPixel a 5 V se evaluó usar MOSFET o 74HC; se optó por 74HCT14 (luego SN74HCT14N) por umbral TTL y Schmitt, además de disponibilidad DIP. Se añadió R 300–500 Ω en DIN y 1000 µF en 5 V (recomendación WS2812), mitigando glitches.

**Firmware v1→v3 (semanas 5–6).**

* La primera versión era bloqueante; migró a loops con ventanas temporales y lectura no bloqueante. Se incorporó EMA para baseline del SHT31. El inductivo pasó a active-low para evitar confusiones. Se incorporó timeout en apertura de tapa (evita deadlocks si RCWL falla).

**Backend y datos (semana 6).**

* Se diseñó la DB mínima: events, proximity, bins, config. La clave fue config: permitió parametrizar (umbral, tiempos, ΔT/ΔHR) sin recompilar. La API quedó idempotente en operaciones de lectura y simple en escritura.

**UI/UX (semana 7).**

* El dashboard tuvo tres metas: *tiempo-a-insight bajo*, *coherencia visual* (verde/negro, Arial, animaciones suaves) y *operabilidad* (instalable PWA, demo sin backend). Las tarjetas por tacho incorporan badges (OK/Warn/Alert) para lectura de un golpe en escenarios reales (celular en pasillo/taller).

**Validación (semana 8).**

* Se simularon cortes de red y caídas de API: la UI mantiene demo sin bloquear. Se forzaron umbrales de 80–90% para evaluar histéresis de alerta y evitar parpadeo. En medición, el error quedó ≤3% tras corregir mecánica (alineación de celda y apoyo rígido). Se actualizaron textos y documentación para reproducibilidad (calibración, cableado y seguridad).

**Ajuste de BOM y naming (semana 9).**

* Se confirmó SN74HCT14N (equivalente funcional a 74HCT14). Se actualizó documentación y se explicitó el uso de dos puertas para no invertir. Se revisaron diagramas y notas de cableado.

**Lecciones aprendidas.**

* **Mecánica manda:** precisión del HX711 depende más del apoyo que de filtros.
* **Separar roles mejora robustez:** IR como trigger; RCWL sólo abre tapas.
* **Parametrizar desde backend:** esto baja la fricción de iterar y evita firmware “duro”.
* **UX simple > “mil features”:** KPIs y badges bien resueltos valen más que diez gráficos.

**Riesgos futuros y palancas.**

* Ambiente (temperatura/humedad) afecta el “orgánico”; si la casuística lo pide, un IR térmico puntual puede fortalecer la decisión.
* Escala: si hay múltiples estaciones, conviene PostgreSQL + uWSGI/Nginx y autenticación.
* Mantenimiento: servos baratos tienen vida finita; registrar ciclos ayudaría a mantenimiento preventivo.

## **Desarrollo Técnico**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Eco Smart – Firmware ESP32 (3 tachos: Metal / Orgánico / Resto)**

**\* - Sensores: IR (trigger), Inductivo (metal), SHT31 (orgánico ΔT/ΔHR), HX711×3 (peso)**

**\* - Actuación: Stepper (ULN2003) + compuerta (servo por PCA9685), NeoPixel + Buzzer**

**\* - Control: FSM no bloqueante + AccelStepper (homing con endstop)**

**\* - Red/API: GET /api/axis (config eje) + POST /api/deposits (reintentos)**

**\* - Sin LDR (plástico/vidrio), sin cuarto tacho**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <WiFi.h> // Wi-Fi STA**

**#include <HTTPClient.h> // HTTP GET/POST**

**#include <Wire.h> // I2C**

**#include <Adafruit\_SHT31.h> // Temp/Hum SHT31**

**#include <Adafruit\_NeoPixel.h> // WS2812**

**#include <Adafruit\_PWMServoDriver.h> // PCA9685 (servo)**

**#include <HX711.h> // HX711 celdas de carga**

**#include <AccelStepper.h> // Motor paso a paso (no bloqueante)**

**#include <ArduinoJson.h> // Parse JSON robusto**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONFIGURACIÓN DE RED/API \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// << Cambiá por tu Wi-Fi y base URL de tu API >>**

**const char\* WIFI\_SSID = "TU\_SSID";**

**const char\* WIFI\_PASS = "TU\_PASSWORD";**

**String API\_BASE = "http://127.0.0.1:5000"; // Ej: "http://192.168.0.50:5000"**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CONFIGURACIÓN DE UMBRALES Y OPERACIÓN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Capacidad nominal de cada tacho en kg (para % de llenado)**

**const float CAPACITY\_KG = 5.0;**

**// Umbral de alerta (UI / señal)**

**uint8\_t ALERT\_THRESHOLD = 80; // % (editable vía API si quisieras)**

**// Umbrales de clasificación (SIN LDR: sólo metal/orgánico/resto)**

**float T\_DH = 3.0f; // Δ%HR mínimo para considerar orgánico**

**float T\_DT = 1.0f; // Δ°C mínimo para considerar orgánico**

**int T\_MIN\_G= 30; // Δg mínimo para confirmar depósito**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PINOUT ESP32 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// I2C (mantener estos pines del ESP32 para evitar conflictos)**

**const int PIN\_SDA = 21;**

**const int PIN\_SCL = 22;**

**// Sensores digitales**

**const int PIN\_IR\_OBS = 26; // Sensor IR de obstáculo (HIGH=hay objeto)**

**const int PIN\_IND\_MET = 27; // Inductivo NPN nivelado (HIGH=metal)**

**// NeoPixel (DIN → SN74HCT14 → tira 5 V). Si cascadas dos tiras, poné el total.**

**const int PIN\_NEOPIX = 23;**

**const int NEOPIX\_N = 16; // cantidad total de LEDs en la cadena**

**// Buzzer 5 V (activar por NPN; este pin maneja la base del transistor)**

**const int PIN\_BUZZER = 2;**

**// HX711: un SCK común y 3 líneas DT (1 por tacho)**

**const int PIN\_HX\_SCK = 25;**

**const int PIN\_HX\_DT[3] = {34, 35, 36}; // pines sólo-entrada en ESP32**

**// PCA9685 (servo compuerta en CH0)**

**Adafruit\_PWMServoDriver pwm = Adafruit\_PWMServoDriver(0x40);**

**// Servo compuerta: calibrar pulsos según tu servo (50 Hz)**

**const int SERVO\_CH = 0;**

**const int SERVO\_MIN\_US = 500;**

**const int SERVO\_MAX\_US = 2400;**

**const int SERVO\_CLOSED = 0; // grados lógicos compuerta cerrada**

**const int SERVO\_OPEN = 60; // grados lógicos compuerta abierta**

**// ULN2003 (stepper unipolar 6 hilos) → AccelStepper FULL4WIRE requiere orden especial**

**const int PIN\_STP\_IN1 = 18;**

**const int PIN\_STP\_IN2 = 19;**

**const int PIN\_STP\_IN3 = 5;**

**const int PIN\_STP\_IN4 = 17;**

**// Endstop X\_min (NC recomendado → activo-bajo)**

**const int PIN\_ENDSTOP = 16;**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HOMING Y VELOCIDADES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**const int HOME\_BOUNCE\_MM = 3; // retroceso para liberar switch**

**const float HOME\_FEED\_MM\_S = 40.0f; // velocidad homing (mm/s)**

**const float HOME\_KISS\_MM\_S = 20.0f; // acercamiento lento (mm/s)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CINEMÁTICA DEL EJE (EDITABLE) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Se puede cargar desde /api/axis en setup(); estos son defaults “seguros”**

**float stepsPerMm = 2.5f; // pasos por mm (depende correa/polea)**

**float vmax\_mm\_s = 120.0f; // velocidad máx eje (mm/s)**

**float acc\_mm\_s2 = 400.0f; // aceleración (mm/s^2)**

**int pos\_mm\_bins[3] = {0, 120, 240}; // posiciones de centro de tachos**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OBJETOS DE LIBRERÍAS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**Adafruit\_SHT31 sht31 = Adafruit\_SHT31();**

**Adafruit\_NeoPixel pixels(NEOPIX\_N, PIN\_NEOPIX, NEO\_GRB + NEO\_KHZ800);**

**HX711 HX[3]; // 3 canales (uno por tacho)**

**// AccelStepper: FULL4WIRE → IN1, IN3, IN2, IN4 (¡orden importante!)**

**AccelStepper stepper(AccelStepper::FULL4WIRE, PIN\_STP\_IN1, PIN\_STP\_IN3, PIN\_STP\_IN2, PIN\_STP\_IN4);**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ESTADO DEL SISTEMA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Máquina de estados: ciclo completo por depósito**

**enum State { IDLE, ENTRY\_SAMPLE, AIM, RELEASE, WEIGHING, RESET };**

**State state = IDLE;**

**// Baselines de SHT31 para detectar “orgánico” por ΔT/ΔHR**

**float baseT = NAN, baseH = NAN;**

**// Calibración HX711 (reemplazar tras calibración real)**

**float hxOffset[3] = {0, 0, 0};**

**float hxScale [3] = {1, 1, 1}; // gramos = (raw - offset)/scale**

**// Tiempos y buffers de la lógica**

**unsigned long tWinStart = 0; // inicio de ventana de clasificación**

**long g0 = 0; // peso antes de soltar compuerta**

**String pendingEventJSON = ""; // buffer RAM de un POST fallido**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* UTILIDADES UI \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Setea todos los NeoPixel con un color**

**void setPixels(uint8\_t r,uint8\_t g,uint8\_t b){**

**for(int i=0;i<NEOPIX\_N;i++) pixels.setPixelColor(i, pixels.Color(r,g,b));**

**pixels.show();**

**}**

**// Beep simple de buzzer**

**void beep(int ms=80){**

**digitalWrite(PIN\_BUZZER, HIGH);**

**delay(ms);**

**digitalWrite(PIN\_BUZZER, LOW);**

**}**

**// Convierte microsegundos a ticks del PCA9685 (50 Hz)**

**uint16\_t usToTicks(int us){**

**float tick = (us/20000.0f) \* 4096.0f;**

**if(tick<0) tick=0; if(tick>4095) tick=4095;**

**return (uint16\_t)tick;**

**}**

**// Escribe “grados lógicos” al servo compuerta**

**void servoWriteDeg(int deg){**

**deg = constrain(deg, 0, 180);**

**int us = map(deg, 0, 180, SERVO\_MIN\_US, SERVO\_MAX\_US);**

**pwm.setPWM(SERVO\_CH, 0, usToTicks(us));**

**}**

**// Helpers abrir/cerrar compuerta**

**void gateOpen(){ servoWriteDeg(SERVO\_OPEN); }**

**void gateClose(){ servoWriteDeg(SERVO\_CLOSED);}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RED / HTTP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Asegurar Wi-Fi conectada (intento rápido)**

**bool wifiEnsure(){**

**if(WiFi.status()==WL\_CONNECTED) return true;**

**WiFi.begin(WIFI\_SSID, WIFI\_PASS);**

**unsigned long t0=millis();**

**while(WiFi.status()!=WL\_CONNECTED && millis()-t0<7000) delay(120);**

**return WiFi.status()==WL\_CONNECTED;**

**}**

**// GET JSON → devuelve string si 200 OK**

**bool httpGetJSON(const String& path, String& out){**

**if(!wifiEnsure()) return false;**

**HTTPClient http; http.begin(API\_BASE + path);**

**int code = http.GET();**

**if(code==200){ out=http.getString(); http.end(); return true; }**

**http.end(); return false;**

**}**

**// POST JSON con 3 reintentos y backoff 250/500/1000 ms**

**bool httpPostJSON\_withRetry(const String& path, const String& json){**

**if(!wifiEnsure()) return false;**

**HTTPClient http; int tries=0; int code=-1;**

**while(tries<3){**

**http.begin(API\_BASE + path);**

**http.addHeader("Content-Type","application/json");**

**code = http.POST(json);**

**http.end();**

**if(code>=200 && code<300) return true;**

**delay( (1<<tries) \* 250 ); // 250, 500, 1000 ms**

**tries++;**

**}**

**return false;**

**}**

**// (Opcional) Enviar estado del eje al backend (para UI en vivo)**

**void pushAxisState(const char\* st, bool homed, float pos\_mm){**

**if (WiFi.status()!=WL\_CONNECTED) return;**

**HTTPClient http;**

**http.begin(String(API\_BASE) + "/api/axis/state");**

**http.addHeader("Content-Type","application/json");**

**String body = String("{\"state\":\"")+st+"\",\"homed\":"+(homed?"true":"false")+**

**",\"pos\_mm\":"+String(pos\_mm,1)+"}";**

**http.POST(body);**

**http.end();**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LECTURAS / SENSORES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Lee HX711 canal i y devuelve gramos promediados**

**long hxReadG(int i, int samples=12){**

**if(!HX[i].is\_ready()) return 0;**

**long sum=0;**

**for(int k=0;k<samples;k++){ sum += HX[i].read(); }**

**long raw = sum/samples;**

**float g = (raw - hxOffset[i]) / hxScale[i];**

**if(g<0) g=0;**

**return (long)g;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CLASIFICACIÓN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Conjunto reducido a 3 materiales**

**enum Material { MAT\_RESTO, MAT\_METAL, MAT\_ORG };**

**// Nombre amigable para registrar/enviar a API**

**const char\* matName(Material m){**

**switch(m){**

**case MAT\_METAL: return "Metal";**

**case MAT\_ORG: return "Orgánico";**

**default: return "Resto";**

**}**

**}**

**// Mapeo de material → índice de tacho (1..3)**

**int binFor(Material m){**

**if(m==MAT\_METAL) return 1;**

**if(m==MAT\_ORG) return 2;**

**return 3; // Resto**

**}**

**// Toma una “muestra” de clasificación (rápida)**

**Material classifyOnce(){**

**// 1) Metal por inductivo NPN (digital)**

**if(digitalRead(PIN\_IND\_MET)==HIGH) return MAT\_METAL;**

**// 2) Orgánico por deltas respecto a baseline exponencial**

**float t = sht31.readTemperature();**

**float h = sht31.readHumidity();**

**if(isnan(baseT)||isnan(baseH)){ baseT=t; baseH=h; } // inicializa baseline**

**float dT = t - baseT;**

**float dH = h - baseH;**

**if(dH >= T\_DH || dT >= T\_DT) return MAT\_ORG;**

**// 3) Si ninguna condición se cumple → Resto**

**return MAT\_RESTO;**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* EJE / STEPPER \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Ajusta kinemática de AccelStepper con pasos/mm y lims**

**inline void axisApplyKinematics(){**

**stepper.setMaxSpeed( vmax\_mm\_s \* stepsPerMm ); // pasos/seg**

**stepper.setAcceleration( acc\_mm\_s2 \* stepsPerMm ); // pasos/seg^2**

**}**

**// Conversión mm ↔ pasos**

**inline long mmToSteps(float mm){ return lround(mm \* stepsPerMm); }**

**inline float stepsToMm(long st){ return (float)st / stepsPerMm; }**

**// Ordena un movimiento no bloqueante a una posición en mm**

**void axisMoveToMM(float mmTarget){**

**stepper.moveTo( mmToSteps(mmTarget) );**

**}**

**// Lee endstop (NC: LOW=presionado)**

**bool endstopHit(){ return digitalRead(PIN\_ENDSTOP)==LOW; }**

**// Rutina de homing a X\_min: rápido → bounce → “kiss” lento → cero**

**void performHoming(){**

**// Guardar velocidades actuales para restaurar al final**

**float origMax = stepper.maxSpeed();**

**float origAcc = stepper.acceleration();**

**// (1) Avance hacia -X hasta tocar endstop**

**stepper.setMaxSpeed(HOME\_FEED\_MM\_S \* stepsPerMm);**

**stepper.setAcceleration((HOME\_FEED\_MM\_S \* stepsPerMm) \* 2);**

**stepper.moveTo( stepper.currentPosition() - 100000 ); // meta muy negativa**

**while(!endstopHit()){ stepper.run(); delay(1); }**

**stepper.stop(); while(stepper.isRunning()) stepper.run();**

**// (2) Retira para liberar y dejar el switch sin presión**

**stepper.setCurrentPosition(0);**

**stepper.moveTo( mmToSteps(HOME\_BOUNCE\_MM) );**

**while(stepper.distanceToGo()!=0){ stepper.run(); delay(1); }**

**// (3) Acercamiento lento (“kiss”) para precisión**

**stepper.setMaxSpeed(HOME\_KISS\_MM\_S \* stepsPerMm);**

**stepper.setAcceleration((HOME\_KISS\_MM\_S \* stepsPerMm) \* 2);**

**stepper.moveTo( -mmToSteps(HOME\_BOUNCE\_MM + 2) );**

**while(!endstopHit()){ stepper.run(); delay(1); }**

**stepper.stop(); while(stepper.isRunning()) stepper.run();**

**// (4) Cero lógico y moverse al centro del tacho 1**

**stepper.setCurrentPosition(0);**

**stepper.setMaxSpeed(origMax);**

**stepper.setAcceleration(origAcc);**

**axisMoveToMM(pos\_mm\_bins[0]);**

**while(stepper.distanceToGo()!=0){ stepper.run(); delay(1); }**

**// Publicar estado “homed” (opcional, si usás en el dashboard)**

**pushAxisState("HOMED", true, stepsToMm(stepper.currentPosition()));**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SETUP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void setup(){**

**Serial.begin(115200);**

**// GPIO básicos**

**pinMode(PIN\_IR\_OBS, INPUT);**

**pinMode(PIN\_IND\_MET, INPUT);**

**pinMode(PIN\_BUZZER, OUTPUT); digitalWrite(PIN\_BUZZER, LOW);**

**pinMode(PIN\_ENDSTOP, INPUT\_PULLUP); // NC: LOW=presionado**

**// NeoPixel (feedback visual)**

**pixels.begin(); setPixels(0, 30, 0); // verde suave “init”**

**// I2C: SHT31 (0x44) + PCA9685 (0x40)**

**Wire.begin(PIN\_SDA, PIN\_SCL);**

**sht31.begin(0x44);**

**pwm.begin(); pwm.setPWMFreq(50);**

**gateClose(); // compuerta cerrada al iniciar**

**// HX711 (3 canales)**

**for(int i=0;i<3;i++){**

**HX[i].begin(PIN\_HX\_DT[i], PIN\_HX\_SCK);**

**HX[i].set\_gain(128);**

**delay(50);**

**}**

**// Stepper: aplicar cinemática inicial**

**axisApplyKinematics();**

**// Wi-Fi**

**WiFi.mode(WIFI\_STA);**

**wifiEnsure();**

**// (Opcional) Cargar kinemática/posiciones desde /api/axis**

**// Espera JSON: {"steps\_per\_mm":..,"vmax":..,"acc":..,"positions\_mm":[...]}**

**String payload;**

**if(httpGetJSON("/api/axis", payload)){**

**StaticJsonDocument<1024> doc;**

**DeserializationError err = deserializeJson(doc, payload);**

**if(!err){**

**if(doc.containsKey("steps\_per\_mm")) stepsPerMm = doc["steps\_per\_mm"].as<float>();**

**if(doc.containsKey("vmax")) vmax\_mm\_s = doc["vmax"].as<float>();**

**if(doc.containsKey("acc")) acc\_mm\_s2 = doc["acc"].as<float>();**

**if(doc.containsKey("positions\_mm") && doc["positions\_mm"].is<JsonArray>()){**

**JsonArray arr = doc["positions\_mm"].as<JsonArray>();**

**int i=0; for (JsonVariant v : arr) { if(i<3) pos\_mm\_bins[i++] = v.as<int>(); }**

**}**

**axisApplyKinematics();**

**}**

**}**

**// Homing al inicio (define cero y posiciona en tacho 1)**

**performHoming();**

**setPixels(0, 80, 20); // verde OK**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LOOP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Llamar siempre run() para mantener stepper no bloqueante**

**inline void pumpMotion(){ stepper.run(); }**

**// Intenta enviar un POST fallido almacenado**

**void tryFlushPendingEvent(){**

**if(pendingEventJSON.length()==0) return;**

**if(httpPostJSON\_withRetry("/api/deposits", pendingEventJSON)){**

**pendingEventJSON = "";**

**}**

**}**

**void loop(){**

**pumpMotion(); // stepper no bloqueante en todas las vueltas**

**static Material lastMat = MAT\_RESTO; // última clase detectada**

**static int targetBin = 3; // tacho objetivo (1..3)**

**switch(state){**

**case IDLE: {**

**// Mantener baseline SHT31 con EMA muy lento (explicabilidad)**

**float t = sht31.readTemperature();**

**float h = sht31.readHumidity();**

**if(!isnan(t)&&!isnan(h)){**

**if(isnan(baseT)||isnan(baseH)){ baseT=t; baseH=h; }**

**else { baseT = baseT\*0.995f + t\*0.005f; baseH = baseH\*0.995f + h\*0.005f; }**

**}**

**setPixels(0, 40, 10); // verde “reposo”**

**tryFlushPendingEvent(); // si hay POST pendiente, reintenta**

**// Disparador: IR detecta objeto**

**if(digitalRead(PIN\_IR\_OBS)==HIGH){**

**tWinStart = millis();**

**lastMat = MAT\_RESTO; // default si nada se decide**

**state = ENTRY\_SAMPLE;**

**setPixels(20, 60, 0); // feedback “sampling”**

**}**

**} break;**

**case ENTRY\_SAMPLE: {**

**// Ventana de muestra (~500 ms): prioridad Metal > Orgánico > Resto**

**if(millis()-tWinStart < 500){**

**Material m = classifyOnce();**

**if(m==MAT\_METAL) lastMat = MAT\_METAL; // prioridad máxima**

**else if(m==MAT\_ORG && lastMat!=MAT\_METAL) lastMat = MAT\_ORG;**

**// Resto no pisa si ya hay algo “mejor”**

**}else{**

**// Elegir tacho final y mandar a esa posición (no bloqueante)**

**targetBin = binFor(lastMat);**

**axisMoveToMM( (float)pos\_mm\_bins[targetBin-1] );**

**pushAxisState("MOVING", true, stepsToMm(stepper.currentPosition())); // opcional**

**state = AIM;**

**}**

**} break;**

**case AIM: {**

**setPixels(40, 80, 0); // “apuntando”**

**// Cuando llega a destino → siguiente estado**

**if(stepper.distanceToGo()==0){**

**pushAxisState("AT\_BIN", true, stepsToMm(stepper.currentPosition())); // opcional**

**state = RELEASE;**

**}**

**} break;**

**case RELEASE: {**

**// Abre compuerta, toma peso antes (g0) y cierra**

**gateOpen();**

**delay(250);**

**gateClose();**

**g0 = hxReadG(targetBin-1, 15);**

**state = WEIGHING;**

**} break;**

**case WEIGHING: {**

**// Pequeña espera de asentamiento mecánico (300 ms)**

**static unsigned long t0 = 0;**

**if(t0==0){ t0=millis(); }**

**if(millis()-t0 < 300) break;**

**t0=0;**

**long g1 = hxReadG(targetBin-1, 15);**

**long dg = g1 - g0; if(dg<0) dg=0;**

**float percent = (float)g1/(CAPACITY\_KG\*1000.0f)\*100.0f;**

**int ipct = (int)percent;**

**// Cuerpo JSON del evento para /api/deposits**

**// (El backend calcula históricos y actualiza “bins”)**

**String json = String("{\"bin\":")+targetBin+**

**",\"material\":\""+String(matName(lastMat))+"\","+**

**"\"weight\_grams\":"+dg+","+**

**"\"fill\_percent\":"+ipct+"}";**

**// Intento con reintentos; si falla, guarda en RAM y reintenta en IDLE**

**if(!httpPostJSON\_withRetry("/api/deposits", json)){**

**pendingEventJSON = json;**

**}**

**// Feedback de alerta local si supera umbral**

**if(percent>=ALERT\_THRESHOLD){ setPixels(180,80,0); beep(120); }**

**else { setPixels(0,80,20); }**

**state = RESET;**

**} break;**

**case RESET: {**

**// Volver al “home operativo” (tacho 1) sin bloquear**

**axisMoveToMM( (float)pos\_mm\_bins[0] );**

**pushAxisState("RETURNING", true, stepsToMm(stepper.currentPosition())); // opcional**

**if(stepper.distanceToGo()==0){**

**pushAxisState("IDLE", true, stepsToMm(stepper.currentPosition())); // opcional**

**state = IDLE;**

**}**

**} break;**

**}**

**}**

## **Conclusión**

### ¿Qué se logró?

Fue difícil, nos costó un montón poder encontrar un proyecto que, cumpla nuestras expectativas, respete la temática y que esté a nuestro alcance realizarlo. No queríamos que nuestro proyecto peque de simple, hubiera sido más sencillo hacer un tacho que solo midiera nivel de llenado y poco más, pero quisimos hacer algo que demuestre nuestras capacidades en el ámbito que llevamos estudiando tres años, la informática. Además, la idea era ver cómo resolver el verdadero problema del reciclaje: el *no* reciclaje. Poca gente es la que se toma el tiempo de revisar donde tira su basura, si es una botella de plástico, o una manzana. Por lo tanto, nuestro proyecto apuntó a resolver *ese* problema, el *verdadero* problema. Cosa que conseguimos, gracias también a los profesores, pudimos encontrar una solución con el sistema de tacos que se mueven con el motor paso a paso.

Al final, logramos demostrar que, a pesar del tiempo y la dificultad de la temática, trabajando bien y en equipo se puede llegar a realizar algo bien, y a tiempo.

### ¿Qué no pudo lograrse?

Hubo montones de ideas, algunas que solo fueron soltadas al aire y no se exploraron lo suficiente como para ser dignas de anotarse. Pensamos enfocarnos en una página más interactiva, siendo lo físico algo más secundario, por ejemplo. Pero todo esto resultaba poco práctico de realizar, lo *más* importante era lo *físico*.

Ya decididos a enfocarnos en los tachos, surgió un problema importante. **No había forma de detectar plásticos.** En vez de eso, tuvimos que optar por detectar el nivel de transparencia/luminosidad, de esta manera, detectando si un objeto es transparente, podemos asumir que probablemente se trate de un plástico o un vidrio. Después vimos que por tiempo y dificultad no podríamos integrar el LDR, ya que lo único que haría ese tacho era ocupar espacio, no tenía un funcionamiento del todo útil y necesario. Así que optamos por descartarlo y dejar solamente tres tachos (orgánicos, metales y basura). Igualmente, tampoco podemos diferenciar el papel, cartón o tela.

### ¿Por qué no pudo lograrse?

El problema radica, no en nuestra capacidad o tiempo, si no en los recursos. No contamos con un sensor que pueda detectar ninguno de esos tipos de materiales, no existe. Por lo tanto, hubo que optar por juntar los objetos transparentes, y dejar de lado el papel/cartón, y demás como otros.

### ¿Cómo podría mejorarse?

**Corto plazo (rápidas y de alto impacto)**

* **Alertas multicanal:** además del dashboard, mandar notificaciones por WhatsApp/email cuando un tacho supere el 80%.
* **Histórico con filtros:** gráfico por día/semana/mes, exportación CSV/PNG y comparativas entre tachos.
* **Panel de servicio:** botones en la web para “Hacer homing”, “Mover a Tacho 1/2/3”, “Probar compuerta” y “Probar buzzer/LED”.
* **Checklist guiado en la UI:** paso a paso para calibrar las celdas de carga y guardar offsets/escala sin reflashear.
* **Modo kiosco:** una vista pública (sólo lectura) para mostrar avances y fomentar reciclaje.

**Mediano plazo (mejoras de producto)**

* **Gamificación:** ranking por cursos/áreas, metas semanales (“menos restos, más reciclado”), medallas y pantallas públicas.
* **Reportes automáticos:** envío mensual con KPIs (kilos recolectados, alertas atendidas, tacho que más se llena).
* **Rutas sugeridas:** priorización de vaciado según ocupación y frecuencia (reglas simples, sin IA).
* **Dashboard multicentro:** ver varios Eco Smart en un mismo mapa (sedes/edificios) y filtrar por ubicación.
* **Ajustes desde la nube:** editar umbrales, posiciones y pasos/mm desde la UI y aplicar a múltiples equipos.

**Largo plazo (evolución más dedicada)**

* **Volver al cuarto tacho (Plástico/Vidrio):** si hace falta, reintroducirlo con mejor encapsulado del sensor óptico o usando cámara+IA más adelante.
* **Tapa automática por tacho:** sensores de proximidad + servos en cada tapa (más vistoso; requiere buena alimentación y mantenimiento).
* **Predicción y planificación:** estimar cuándo se llenará cada tacho (tendencias) y generar planes de vaciado.
* **Integración con BI:** enviar datos a Grafana/Power BI/Metabase para tableros ejecutivos.
* **Mapa de ocupación:** calor por zonas; ayuda a decidir dónde reubicar tachos.

**Robustece el sistema (confiabilidad y mantenimiento)**

* **Copia local de eventos:** si se corta internet, guardar y reenviar luego (ya lo hace básico; ampliarlo).
* **Auto-homing programado:** homing diario o por horas de baja actividad para mantener precisión.
* **Autodiagnóstico:** alertas si un sensor se desconecta, si el motor pierde pasos, si una celda deriva.
* **Logs y salud:** página “/health”, contadores de errores y tiempo en línea.

**Experiencia de usuario y comunicación**

* **Microtutoriales en la web:** tips breves (“Cómo calibrar”, “Qué hacer si no hay Wi-Fi”).
* **Modo educativo:** datos en tiempo real + equivalencias (kilos → árboles/energía) para concientizar.
* **Identidad visual:** theming más pulido, animaciones suaves, stickers/QR en los tachos con el estado.

**Sostenibilidad y costos**

* **Ahorro energético:** bajar brillo de NeoPixel, dormir Wi-Fi fuera de horario, y “reposo” del motor.
* **Mantenimiento fácil:** modularizar el cableado (conectores) y dejar repuestos básicos (servos, celdas, buck).

Si tuviéramos que priorizar: primero pondríamos las alertas multicanal más los reportes, luego añadiríamos los histórico con filtros y panel de servicio, despúes agregamos la gamificación básica, y por último estaría guardar eventos offline y autodiagnóstico. Con eso, el proyecto “crece” de prototipo a sistema usable día a día, y después sí vale la pena encarar tapas automáticas o un cuarto tacho.

Créditos y autoría

Desarrollado y documentado por **Manfredo, Ferrando, Iannone, Siciliano y Montenegro**, con foco en **informática, robótica y electrónica aplicada** para gestión de residuos.