
**UNIVERSIDAD TÉCNICA DE MACHALA**

# Documentación del Proyecto

*Sistema de Parqueadero Inteligente con Facturación Automática*

**Autores:**

Castillo Ortega Brandon Estefano
Peña Paladines Martin Alexander

**Asignatura:**

Internet de las Cosas

**Fecha:**

9 de junio de 2025

# Índice

# 1. Arquitectura General del Sistema

El sistema se fundamenta en una arquitectura de flujos en Node-RED que procesa eventos en tiempo real. La lógica se divide en tres dominios principales:

- **Recepción de Datos (MQTT):** Suscripción a los tópicos de los sensores de cada slot.
- **Procesamiento Central y Lógica de Negocio:** Un nodo de función monolítico que gestiona el estado, la lógica de facturación y la preparación de datos para la UI.
- **Interfaces de Usuario:** Un dashboard web para visualización y un bot de Telegram para interacción y reportes.

# 2. Gestión de Estado

El sistema utiliza dos niveles de variables de contexto para mantener la persistencia de los datos durante la ejecución.

- **Contexto de Flujo (‘flow‘):** Se usa para almacenar el estado actual de los 12 slots (‘flow.parkingStatus‘). Esta variable contiene un objeto donde cada clave es el número del slot y su valor incluye el estado (’Ocupado’/’Libre’) y la hora de entrada (‘entryTime‘). Se eligió el contexto de flujo para que los datos estén aislados a este tablero específico.
- **Contexto Global (‘global‘):** Se emplea para guardar el historial de todas las transacciones de facturación (‘global.allBillings‘). Se optó por el contexto global para que este registro sea potencialmente accesible desde otros flujos o sistemas en el futuro (ej. un flujo de contabilidad separado).
- **Inicialización:** Un nodo `inject`, configurado para ejecutarse una vez al desplegar, invoca una función que limpia ambas variables de contexto (‘flow.set(’parkingStatus’, {})‘ y ‘global.set(’allBillings’, [])‘). Esto garantiza un arranque limpio y previene la retención de datos de sesiones anteriores.

# 3. Lógica de Negocio: Facturación Automática

El núcleo de la lógica de negocio se encuentra en el nodo de función **.^ctualizar Estado y UI”**, que posee dos salidas para separar la actualización de la UI del evento de facturación.

## 3.1. Detección de Eventos y Temporización

La lógica se activa únicamente cuando el estado de un slot cambia (‘newStatus !== oldStatus‘) para optimizar el procesamiento.

- **Inicio de Estadía:** Cuando un slot pasa a ’Ocupado’, el sistema registra la hora de entrada (‘entryTime: new Date()‘).
- **Fin de Estadía y Cálculo:** Cuando el estado cambia de ’Ocupado’ a ’Libre’, se calcula la duración en milisegundos. Para propósitos de prueba, la duración se convierte a ”horas de prueba”donde cada segundo equivale a una hora.

```
// Cuando el slot es liberado
const entryTime = new Date(oldSlotData.entryTime);
const exitTime = new Date();
const durationMs = exitTime.getTime() - entryTime.getTime();

// MODO PRUEBA: 1 segundo = 1 hora
const durationHours = durationMs / 1000;

// L gica de cobro (m nimo 1 hora)
if (durationHours >= 1) {
    const billableHours = Math.floor(durationHours);
    const ratePerHour = 2; // Tarifa por hora
    const cost = billableHours * ratePerHour;
    // ... preparar mensaje de facturaci n
}
```

Listing 1: Cálculo de duración y costo.

### 3.2. Generación de Eventos de Facturación

Si la estadía cumple con el mínimo para ser facturable, se construye un objeto `msg2` para la segunda salida del nodo.

- Este objeto contiene un `type: 'billing_event'` para que el dashboard lo pueda identificar y procesar de forma diferenciada.
- Incluye todos los datos relevantes: `slot`, `duration`, `cost`, y `billableHours`.
- Este mismo objeto se añade al array 'global.allBillings' para mantener el registro histórico.

## 4. Procesamiento de Comandos de Telegram

La interacción con el bot se gestiona mediante una cadena de nodos que enrutan y procesan comandos de texto.

### 4.1. Enrutamiento de Comandos

Un nodo `switch` utiliza expresiones regulares (RegExp) para identificar el comando enviado por el usuario y dirigir el flujo a la lógica apropiada.

- '/*status*' : *Coincideconmensajesqueempiezancon*/*status*.'/*slot*' : *Coincideconmensajesqueempiezancon*/*slot*.
- '/*set*' : *Coincideconmensajesqueempiezancon*/*set*.'/*reporte*' : *Coincideconmensajesqueempiezancon*/*reporte*.

### 4.2. Lógica de Comandos Específicos

- **Reporte de Slot Individual:** Utiliza 'command.match(/
  /slot(
  d+)/)' para extraer el número del slot directamente del texto del comando.
- **Simulación de Sensor:** El comando `/set` es procesado por una función que genera dos salidas: una simula un mensaje MQTT para actualizar el estado global del sistema, y la otra envía una confirmación al usuario de Telegram.
- **Reporte Diario:** El nodo "Generar Reporte Diariorecupera el historial de 'global.allBillings', filtra las transacciones cuya fecha coincida con la del día actual, y calcula el total. Esto demuestra una separación clara entre la recolección de datos y la generación de informes.

# 5. Capturas del Sistema

## 5.1. Flujo General en Node-RED

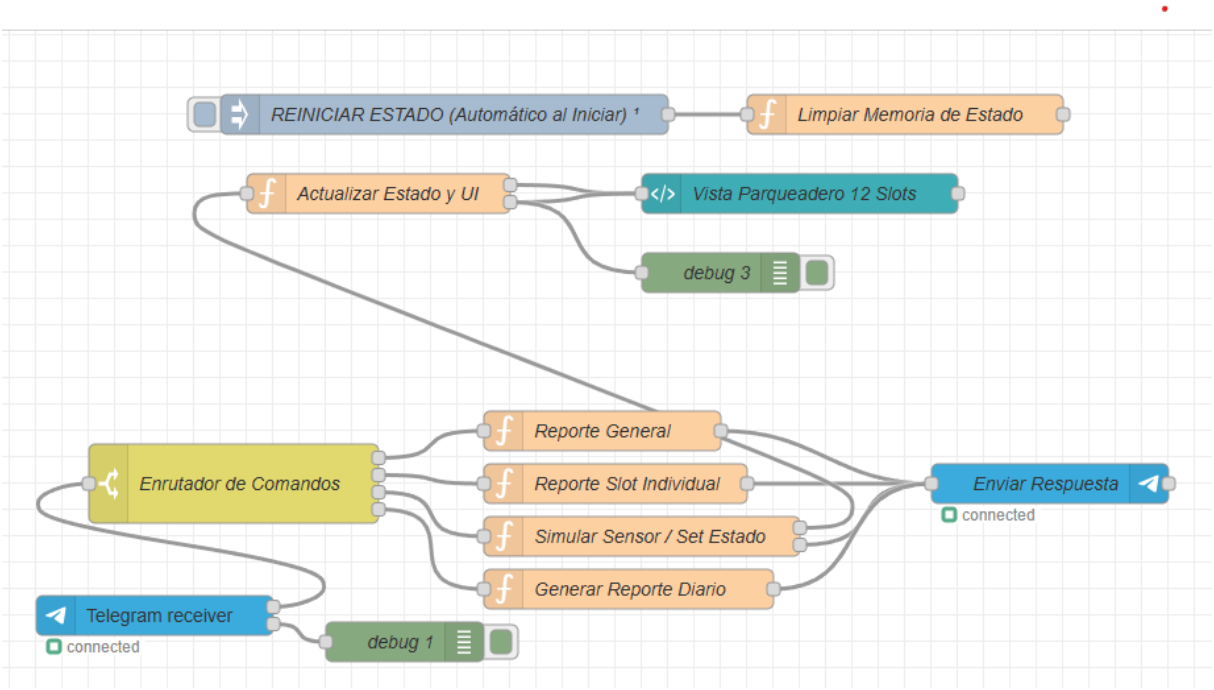


Figura 1: Vista del flujo final.

## 5.2. Dashboard con Sección de Cobros

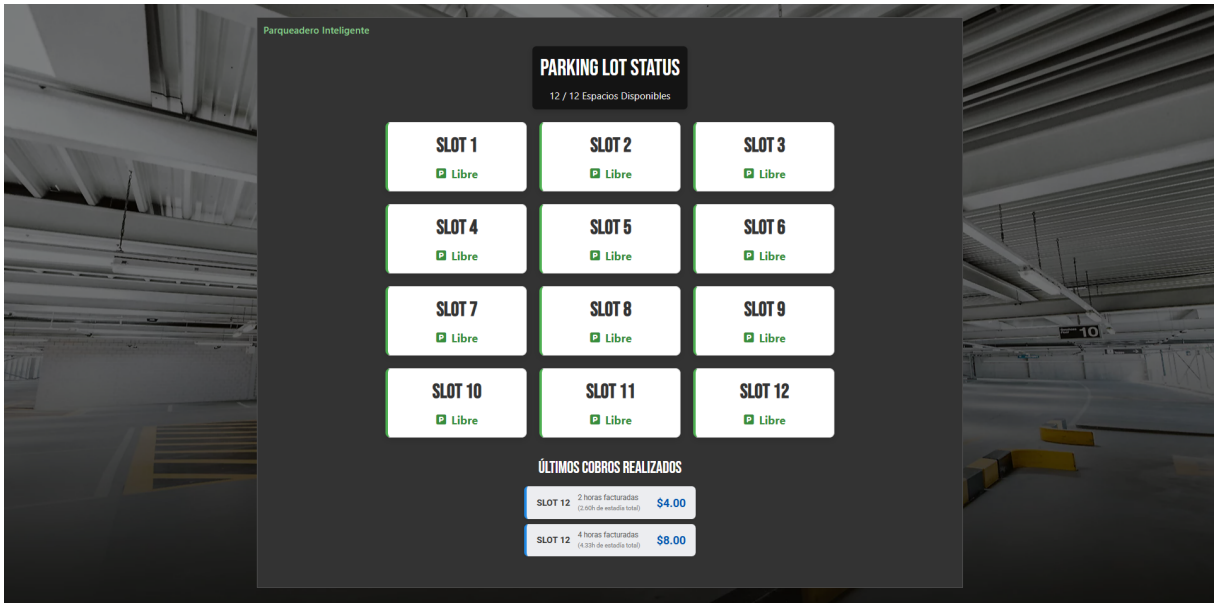


Figura 2: Dashboard mostrando la lista de cobros.

## 5.3. Lógica de Facturación en el Nodo Principal

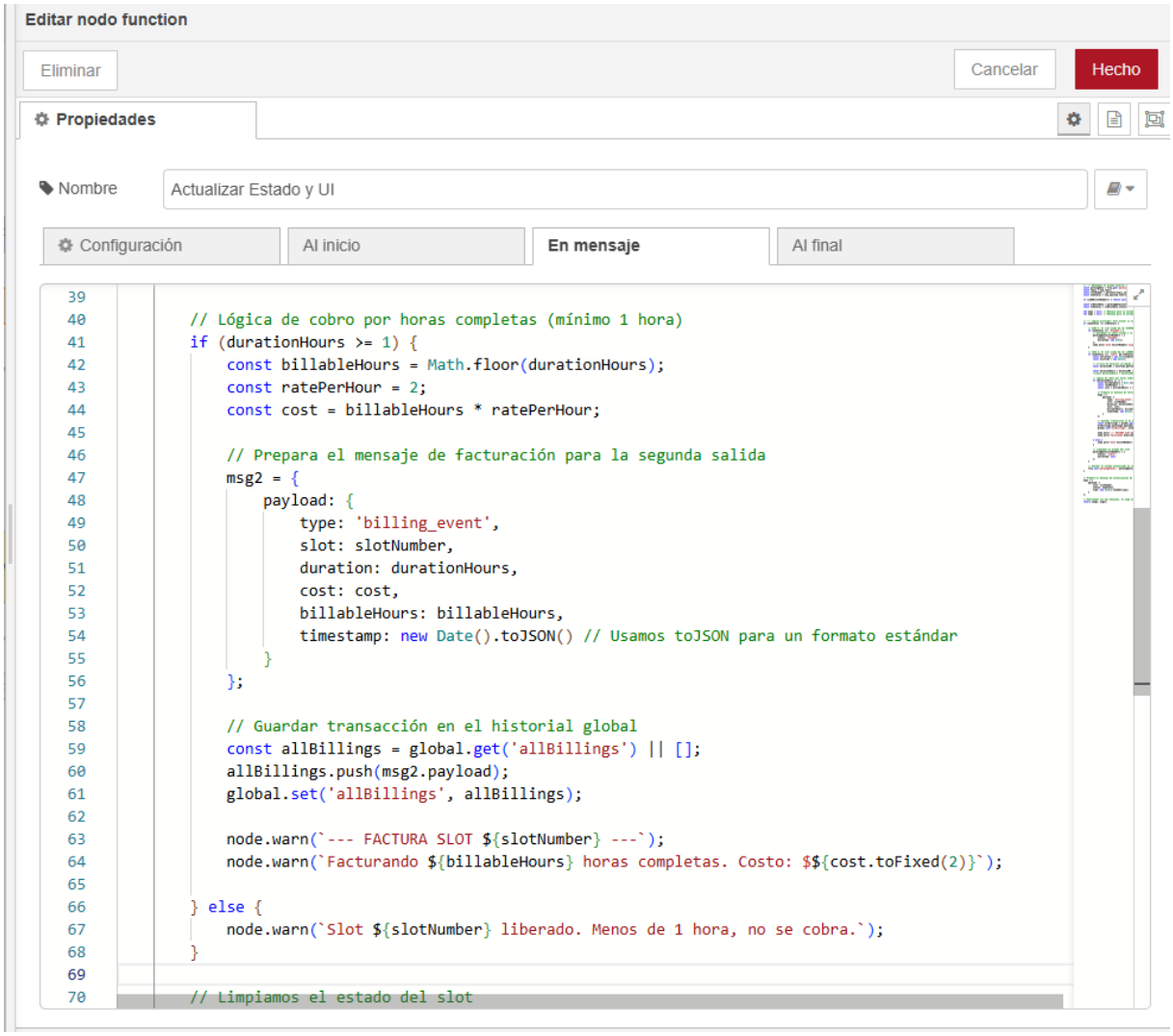


Figura 3: Detalle del código del nodo .ªctualizar Estado y UI".

### 5.4. Bot de Telegram con Comando de Reporte

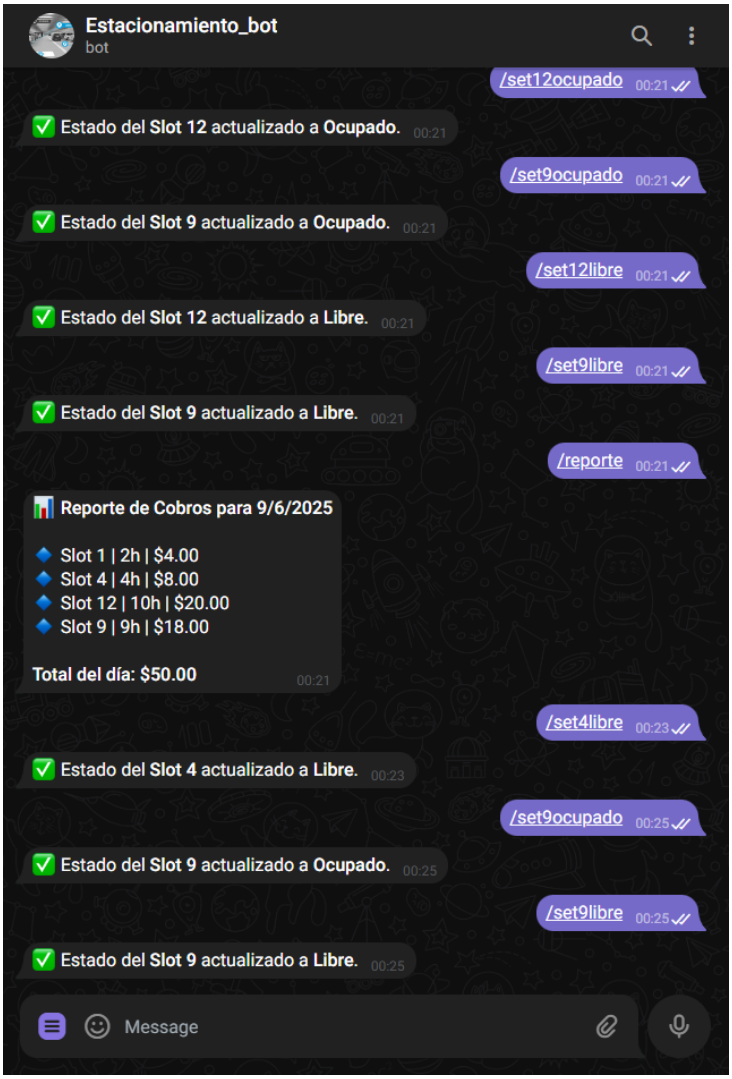


Figura 4: Ejemplo de uso del comando `/reporte`.

## 6. Conclusiones

La implementación de este proyecto demuestra la alta capacidad de Node-RED como herramienta de orquestación para soluciones complejas de IoT. El sistema no solo cumple con los objetivos de monitoreo visual a través de un dashboard web, sino que lo trasciende al integrar un bot de Telegram para la interacción remota y, fundamentalmente, un sistema de facturación automática.

La lógica de negocio para el cálculo de costos basado en el tiempo de estadía, junto con la persistencia de datos de transacciones y la generación de reportes diarios, eleva el proyecto de un prototipo funcional a una aplicación con claro potencial de negocio. La arquitectura, que separa la actualización de la UI de los eventos de facturación y centraliza la lógica de estado, ha demostrado ser robusta y escalable.

Como trabajo futuro, el sistema podría expandirse para incluir bases de datos más robustas, sistemas de autenticación de usuarios y la implementación de una plataforma de pago en línea.

# 7. Bibliografía

## Referencias


[1] "Node-red-dashboard," *Nodered.org.* [En línea]. Disponible: https://flows.nodered.org/node/node-red-dashboard.

[2] "node-red-contrib-telegrambot," *Nodered.org.* [En línea]. Disponible: https://flows.nodered.org/node/node-red-contrib-telegrambot.

[3] "HiveMQ Public MQTT Broker," *Hivemq.com.* [En línea]. Disponible: https://www.hivemq.com/public-mqtt-broker/.

[4] "Node-RED Documentation," *Nodered.org.* [En línea]. Disponible: https://nodered.org/docs/.