

```

/**=
=====
=====

* 01_Principal.gs - Núcleo de sincronización (ICS ↔ Resumen ↔ Google
Calendar)

* Reglas confirmadas:
*   - Ventana: hoy → +30 días (nunca pasado)
*   - ICS manda (prioridad sobre Form/OMITIR en conflicto)
*   - Eventos all-day; "Fecha servicio" = día de salida (DTEND - 1 si
aplica)
*   - RESUMEN_TEST es snapshot limpio de 30 días en cada pasada iCal
*   - Watcher nocturno (03:05) relee TODO el rango de 30 días y refleja
cambios en OBS/Fecha/Estado
*   - Creación/actualización en Calendar: ahora se hace vía Webhook +
Make.com.
*   (La función pushResumenToCalendars directa a Calendar se mantiene
solo como legacy/emergencia.) *   - Si piso no existe en BASE_DATOS →
Estado=ERROR y en OBS "piso no encontrado en Base_Datos"
===== */

```

```

/** Limpia triggers de menús legacy (una vez) */
function __cleanupLegacyOnOpenTriggers(){
  const legacy = new Set([
    'buildMenuResumenMensual', 'buildMenuResumenUtils', '__onOpen_menu_cierre',
    'addMenuFacturacion', 'onOpenOld', '__onOpen_menu_resumen'
  ]);
  const all = ScriptApp.getProjectTriggers();
  let removed = 0;
  all.forEach(t=>{
    const h = t.getHandlerFunction ? t.getHandlerFunction() : '';
    if (legacy.has(h)) { ScriptApp.deleteTrigger(t); removed++; }
  });
}
```

```
});

Logger.log('Triggers legacy eliminados: ' + removed);
SpreadsheetApp.getActive().toast('Limpieza realizada. Reabre el archivo.',
'Operaciones', 5);

}

/*
=====
LISTA GLOBAL DE CALENDARIOS (con guarda)
=====
if (typeof CALENDARIOS === 'undefined') var CALENDARIOS = {
  "ABB Turistic Aparments SL": "bf5296962d9e332e0f51466b7550a560be80d161d30ea9d0ce40c02f6ea82c5e@group.calendar.google.com",
  "Andrea Serna Garcia": "9bfdd64a7cc866ab95bbb414d8568b79d963c957226377ba3c12752a746f5801@group.calendar.google.com",
  "BCN Poble Nou XXI SL": "432e34cadd6b5380cf84da37421cf7408e307ad2acb674735cb6ab27a188535d@group.calendar.google.com",
  "Bellesi Euro": "2ea0abb1daeaca353225c4f7cd5df8ac1de8a39556150b28994c42eefc785ae6@group.calendar.google.com",
  "Catalunya Assessors i Taxadors": "44559558704e1930eae880268e08175383327e84bc60a0b4ad68ae8210a69a7c@group.calendar.google.com",
  "EMITE TU LUZ S.L.": "4e2a139f266cb0515a8ea26d9f7f335b66037e2b528888a5ad23c1e865ca5682@group.calendar.google.com",
  "KAWAKAN SL": "44400d0ec0cbdfc2520c4eab480768670c0e20f7180d3302a9afb527c96841bf@group.calendar.google.com",
}
```

```
"NOUGAVA S.A.U." :  
"65295c00757c17710e803b4fec8608af97cf9335577420a7bdb4d85c7751799a@group.cal  
endar.google.com",  
"STAY ON BARCELONA SL" :  
"3ed92e772f473c829366c3996ec7a09dfab8e39c956fb3503eb50a20136e7a5c@group.cal  
endar.google.com",  
"THE AURORA'S FLAT BARCELONA SL" :  
"55e116bb294252fa136e08384307818b332ea26de76c480cc62c97f7ff24d7e7@group.cal  
endar.google.com",  
"XAVIER MELCHOR FERRER" :  
"d98850cd2763008d66623b1841da3b5f2c1f45894041793c63b7182ad4cf6252@group.cal  
endar.google.com",  
"TESTTEST" :  
"cf8bd812b940836861f29a4e1f249de3fd100134d79f199a9b554e7e76b54401@group.cal  
endar.google.com",  
"BARCELONAFORRENT SL" :  
"d4f0e01d2f7dbbe705ebdf89ca6812c99a444e8f1612b380545545c6bb6b1518@group.cal  
endar.google.com"  
};
```

```
/* ======  
CONFIG + HELPERS COMUNES (con guardas)  
===== */  
  
// Config (solo si no existen ya en otro .gs)  
if (typeof TZ === 'undefined') var TZ = 'Europe/Madrid';  
if (typeof SHEET_SOURCES === 'undefined') var SHEET_SOURCES =  
'ICAL_SOURCES'; // PISO | ICS_URL
```

```

if (typeof SHEET_RESUMEN_TEST === 'undefined') var SHEET_RESUMEN_TEST =
'RESUMEN_TEST'; // snapshot 30 días (ICS)
if (typeof SHEET_RESUMEN === 'undefined') var SHEET_RESUMEN = 'Resumen';
if (typeof SHEET_BASE_DATOS === 'undefined') var SHEET_BASE_DATOS =
'BASE_DATOS';

if (typeof IMPORT_FUTURE_DAYS === 'undefined') var IMPORT_FUTURE_DAYS = 30;
if (typeof IMPORT_PAST_DAYS === 'undefined') var IMPORT_PAST_DAYS = 0;
// nunca pasado

// Helpers
function getSheet(name){ return
SpreadsheetApp.getActive().getSheetByName(name); }
function norm(s){ return String(s||'').trim(); }
function _normTxt(s){ return
String(s||'').normalize('NFD').replace(/[\u0300-\u036f]/g,'').toLowerCase()
.replace(/\s+/g,' ').trim(); }
function _normClienteKey(s){
return
_normTxt(s).replace(/\b(s\.\?l\.\?|sau)s\.\a\.\u\.)\b/g,'').replace(/\[^a-z0-9]+
/g,' ').trim();
}
function asLocalDateStr(d){ return Utilities.formatDate(d, TZ,
'dd/MM/yyyy'); }
function asLocalTs(d){ return Utilities.formatDate(d, TZ, 'dd/MM/yyyy
HH:mm:ss'); }
function startOfDay(d){ d.setHours(0,0,0,0); return d; }
function endOfDay(d){ d.setHours(23,59,59,999); return d; }
function toMidday(d){ d.setHours(12,0,0,0); return d; }
function inWindow(date, from, to){ return (date>=from && date<=to); }
function ymd(d){ return Utilities.formatDate(d, TZ, 'yyyy-MM-dd'); }
function addDays(d, n){ const x=new Date(d.getTime());
x.setDate(x.getDate()+n); return x; }

```

```

function keyFrom(piso, fechaStr, tipo){ return
` ${norm(piso)}_${norm(fechaStr)}_${norm(tipo).toLowerCase()}`; }

function sha1(s){ return
Utilities.computeDigest(Utilities.DigestAlgorithm.SHA_1, s).map(b=>(`0'+(b
& 0xFF).toString(16)).slice(-2))).join(''); }

function getCalendarIdForCliente(cliente){
  if (!cliente) return null;
  if (CALENDARIOS[cliente]) return CALENDARIOS[cliente];
  const nk = _normClienteKey(cliente);
  for (const k in CALENDARIOS){ if (_normClienteKey(k) === nk) return
CALENDARIOS[k]; }
  return null;
}

/***
 * Fuerza el formato DD/MM/YYYY en la columna "Fecha" de la hoja RESUMEN
 */
/***
 * Fuerza el formato DD/MM/YYYY en la columna "Fecha" de la hoja RESUMEN
 * y convierte textos tipo "Sat Dec 13 2025..." a fecha real.
 */
function fixFormatoFechaResumen_(){
  const sh = getSheet(SHEET_RESUMEN);
  if (!sh) return;

  const lastRow = sh.getLastRow();
  if (lastRow < 2) return;

  const header = sh.getRange(1, 1, 1, sh.getLastColumn())
    .getValues()[0]
    .map(v => String(v || ''));

  const colFecha = header.indexOf('Fecha');
  if (colFecha === -1) return;

```

```
const range = sh.getRange(2, colFecha + 1, lastRow - 1, 1);
const values = range.getValues();

// 1) Convertir textos "raros" a Date
for (let i = 0; i < values.length; i++) {
    const v = values[i][0];
    if (!v) continue;

    // Si ya es Date, lo dejamos
    if (v instanceof Date && !isNaN(v)) continue;

    // Si es string, intentamos convertirlo
    if (typeof v === 'string') {
        const d = new Date(v);
        if (!isNaN(d)) {
            values[i][0] = d;
        }
    }
}

range.setValues(values);

// 2) Aplicar formato dd/MM/yyyy a toda la columna de fechas
range.setNumberFormat('dd/MM/yyyy');
}

/**
 * Utilidad manual: arreglar todas las fechas de RESUMEN
 */
function fixFechasResumen() {
    fixFormatoFechaResumen_();
}
```

```
/* =====
MAPA BASE_DATOS (caché)
===== */

let __BASE_MAP = null;

function buildBaseMaps_(){

const bd = getSheet(SHEET_BASE_DATOS);

const last = bd.getLastRow(), lastCol = bd.getLastColumn();
const vals = last > 1 ? bd.getRange(2,1,last-1,lastCol).getValues() : [];
const mapByPiso = new Map(); // piso -> {cliente, limpiadora, correo, ...}
vals.forEach(row=>{

    const cliente = row[0], piso = row[6], limpiadora=row[13],
correo=row[14];

    if (piso) mapByPiso.set(String(piso).trim(), {cliente, limpiadora,
correo, __row: row});
});

return mapByPiso;
}

function getBaseMap(){ if (!__BASE_MAP) __BASE_MAP = buildBaseMaps_();
return __BASE_MAP; }

function refreshBaseMap(){ __BASE_MAP = buildBaseMaps_(); }

function obtenerClientePorPiso(piso){ const rec =
getBaseMap().get(String(piso).trim()); return rec ? (rec.cliente ||
"ERROR") : "ERROR"; }

function obtenerInfoLimpiadora(piso){ const rec =
getBaseMap().get(String(piso).trim()); return rec ? [rec.limpiadora||"",
rec.correo||"" : ["","",""]; }
```

```

/*
=====
=====

1) ENTRADAS INMEDIATAS - FORM / OMITIR_*
(Se mantienen como estaban: "al momento")

=====
===== */

function onFormSubmit(e){
  const ss      = e.source;
  const sheet  = e.range ? e.range.getSheet() : ss.getActiveSheet();
  const nombre = sheet.getName();
  const fila   = e.range ? e.range.getRow() : sheet.getLastRow();

  if (nombre === "BASE_DATOS" || nombre === "Resumen") return;
  if (fila < 2) return;

  if (nombre.startsWith("OMITIR_")) {
    separarPisos_(ss, sheet, fila);
    return;
  }

  const rowVals = sheet.getRange(fila, 1, 1,
sheet.getLastColumn()).getValues()[0];
  const [timestamp, pisoIn, fechaServicio] = rowVals;
  const cliente = nombre;

  const basePiso =
String(pisoIn||'').replace(/\s+(intermedia|repaso)$|i,'').trim();
  const filaFinal = [ timestamp, pisoIn, fechaServicio, cliente ];
}

```

```

const resumen = ss.getSheetByName(SHEET_RESUMEN);
resumen.appendRow(filaFinal);

const newRow = resumen.getLastRow();
const [limpiadora, correoLim] = obtenerInfoLimpiadora(basePiso);
resumen.getRange(newRow, 12).setValue(limpiadora); // L
resumen.getRange(newRow, 13).setValue(correoLim); // M

aplicarFormulasResumen(newRow);
setEstadoPendienteResumen_(newRow); // ← NUEVO
notificarWebhook(newRow);
}

/** Parser genérico "OMITIR_*" (igual que tenías) */
function separarPisos_(ss, sheet, fila){
  const nombre = sheet.getName();
  const resumen = ss.getSheetByName(SHEET_RESUMEN);
  const datos = sheet.getRange(fila, 1, 1,
sheet.getLastColumn()).getValues()[0];
  const encabezados = sheet.getRange(1, 1, 1,
sheet.getLastColumn()).getValues()[0];

  if (nombre === "BASE_DATOS" || nombre === "Resumen" || fila < 2) return;

  if (nombre === "OMITIR_11") { procesarOmitir11_(ss, sheet, fila); return;
}

```

```

const timestamp = datos[0];

const idxFechaHeader = encabezados.findIndex(h => /fecha/i.test(h));
const idxFecha       = idxFechaHeader >= 0 ? idxFechaHeader : 2;
const fechaServicio = datos[idxFecha];

let textoCompleto = datos[1];

if (encabezados.some(h => /\[[^\]]+\]/.test(h))) {
  const startCol = idxFecha + 1;
  const parts = [];
  for (let c = startCol; c < encabezados.length; c++) {
    const marcado = datos[c];
    if (!marcado) continue;
    const m = encabezados[c].match(/\[(\w+)\]/);
    let piso = m ? m[1] : encabezados[c];
    const s = marcado.toString().toLowerCase();
    if (s === "intermedia") piso += " intermedia";
    else if (s === "repaso") piso += " repaso";
    else if (s === "check in") piso += " - Check In";
    parts.push(piso);
  }
  textoCompleto = parts.join(", ");
}

textoCompleto.split(", ").map(p => p.trim()).forEach(textoPiso => {
  const idxSep = textoPiso.lastIndexOf("-");
  let piso      = (idxSep > -1) ? textoPiso.slice(0, idxSep).trim() :
  textoPiso;
}

```

```
let servicio = (idxSep > -1) ? textoPiso.slice(idxSep + 1).trim() :
"Limpieza";

const sl = servicio.toLowerCase();
if (sl === "intermedia") piso += " intermedia";
else if (sl === "repaso") piso += " repaso";

const basePiso = piso.replace(/\s+(intermedia|repaso)$/, "");
const cliente = obtenerClientePorPiso(basePiso);
const filaNueva = [ timestamp, piso, fechaServicio, cliente ];

const hojaCli = ss.getSheetByName(cliente);
if (hojaCli) hojaCli.appendRow(filaNueva);

resumen.appendRow(filaNueva);
const newRow = resumen.getLastRow();

const [limpiadora, correoLim] = obtenerInfoLimpiadora(basePiso);
resumen.getRange(newRow, 12).setValue(limpiadora);
resumen.getRange(newRow, 13).setValue(correoLim);

aplicarFormulasResumen(newRow);
setEstadoPendienteResumen_(newRow); // ← NUEVO
notificarWebhook(newRow);
});

}

/** Parser específico OMITIR_11 (igual que tenías) */
```

```

function procesar0mitir11_(ss, sheet, fila) {
  const [timestamp, fechaServicio, textoBruto] = sheet.getRange(fila, 1, 1,
3).getValues()[0];
  const textoPiso = String(textoBruto).trim();
  const tokensUser = _normTxt(textoPiso).split(/\s+/).filter(t => t);

  const bd    = ss.getSheetByName(SHEET_BASE_DATOS);
  const data = bd.getRange("A2:L" + bd.getLastRow()).getValues();

  let matchRow = null;
  for (let i = 0; i < data.length; i++) {
    const row      = data[i];
    const clienteBD = String(row[0]).trim().toUpperCase();
    if (clienteBD !== "NOUGAVA S.A.U.") continue;

    const pisoBD      = String(row[6]).trim(); // G
    const normPisoBD = _normTxt(pisoBD);
    if (tokensUser.every(tok => normPisoBD.includes(tok))) { matchRow = row;
break; }
  }

  if (!matchRow) { sheet.getRange(fila, 4).setValue("ERROR"); return; }

  const matchedText = String(matchRow[6]).trim();
  const parts      = matchedText.split(/\s+/);
  const basePiso   = parts[0];
  const idInterno = parts[1];
  const servicio   = parts.slice(2).join(" ");
  const resultado = `${basePiso} ${idInterno} ${servicio}`;
}

```

```
sheet.getRange(fila, 4).setValue(resultado);

const cliente      = "NOUGAVA S.A.U.";
const filaDatos   = [ timestamp, resultado, fechaServicio, cliente ];

const hojaCli = ss.getSheetByName(cliente);
if (hojaCli) hojaCli.appendRow(filaDatos);

const resumen      = ss.getSheetByName(SHEET_RESUMEN);
resumen.appendRow(filaDatos);
const newRow       = resumen.getLastRow();
const precioLimpia = matchRow[9];    // J
const precioRopa   = matchRow[11];   // L
resumen.getRange(newRow, 5).setValue(precioLimpia);
resumen.getRange(newRow, 6).setValue(precioRopa);
resumen.getRange(newRow, 7).setValue(precioLimpia + precioRopa);

const [limNom, correoLim] = obtenerInfoLimiadora(matchedText);
resumen.getRange(newRow, 12).setValue(limNom);
resumen.getRange(newRow, 13).setValue(correoLim);

setEstadoPendienteResumen_(newRow);    // NUEVO
notificarWebhook(newRow);
}

/***
 * Marca la fila indicada de RESUMEN con Estado = "Pendiente"
 */

```

```

function setEstadoPendienteResumen_(fila) {
  const sh = getSheet(SHEET_RESUMEN);
  if (!sh || fila < 2) return;

  const header = sh.getRange(1, 1, 1, sh.getLastColumn())
    .getValues()[0]
    .map(v => String(v || ''));
  const col = header.indexOf('Estado');
  if (col === -1) return; // por si acaso

  // indexOf es 0-based → sumamos 1 para número de columna
  sh.getRange(fila, col + 1).setValue('Pendiente');
}

/*
=====
Fórmulas Resumen (igual)
===== */
function aplicarFormulasResumen(fila) {
  const hoja =
    SpreadsheetApp.getActiveSpreadsheet().getSheetByName(SHEET_RESUMEN);
  const formulaE = `=IF(B${fila}="" ; "" ; VLOOKUP(B${fila} ; BASE_DATOS!G:J;
4; FALSE))`;
  const formulaF = `=IF(B${fila}="" ; "" ; VLOOKUP(B${fila} ; BASE_DATOS!G:L;
6; FALSE))`;
  const formulaG = `=IF(AND(E${fila}="" ; F${fila}="") ; "" ; IFERROR(E${fila};
0) + IFERROR(F${fila}; 0))`;
  hoja.getRange(`E${fila}`).setFormula(formulaE);
  hoja.getRange(`F${fila}`).setFormula(formulaF);
  hoja.getRange(`G${fila}`).setFormula(formulaG);
}

```

```
/* =====
Webhook Make.com (igual)
===== */

function notificarWebhook(fila) {
  const ss    = SpreadsheetApp.getActive();
  const hoja = ss.getSheetByName(SHEET_RESUMEN);
  if (!hoja) return;

  const row = (typeof fila === 'number' && fila >= 2) ? fila :
  hoja.getLastRow();
  const valores = hoja.getRange(row, 1, 1,
  hoja.getLastColumn()).getValues()[0];

  const payload = {
    row,
    timestamp:      valores[0],
    piso:           valores[1],
    fechaServicio: valores[2],
    cliente:        valores[3],
    limpiadora:     valores[11],
    correoLim:      valores[12],
  };

  const options = { method: "post", contentType: "application/json",
payload: JSON.stringify(payload), muteHttpExceptions: true };
  const webhookUrl =
"https://hook.eu2.make.com/1r2db8mh4kdjnav9ah5ccqjskaq6mhfh";
  try { UrlFetchApp.fetch(webhookUrl, options); } catch (err) {
Logger.log("Webhook error: %s", err); }
```

```
}

/*
=====
=====

2) ICS → RESUMEN_TEST (SNAPSHOT 30 DÍAS, ALL-DAY, UID SINTÉTICO SI FALTA)
3) RESUMEN_TEST → Resumen (ICS PRIORIZA)
===== */
===== */

function syncIcsSnapshotToResumenTest(){
  const shSrc = getSheet(SHEET_SOURCES);
  const shTmp = getSheet(SHEET_RESUMEN_TEST);
  if (!shSrc || !shTmp) throw new Error('Falta ICAL_SOURCES o
RESUMEN_TEST');

  // Ventana [hoy..+30]
  const now = new Date();
  const from = startOfDay(new Date(now));
  const to = endOfDay(new Date(now));
  to.setDate(to.getDate()+IMPORT_FUTURE_DAYS);

  // Cabecera mínima en RESUMEN_TEST
  const header =
['Fecha', 'Tipo', 'Piso', 'Días', 'Estado', 'Clave', 'Origen', 'UID', 'OBS', 'Marca
de Tiempo'];
  shTmp.getRange(1,1,1,header.length).setValues([header]);
}
```

```

// Limpia snapshot (excepto cabecera)
const last = shTmp.getLastRow();
if (last >= 2) shTmp.getRange(2,1,last-1,
shTmp.getLastColumn()).clearContent();

// Lee fuentes: PISO | ICS_URL
const srcLast = shSrc.getLastRow();
if (srcLast < 2) return;

const rows = shSrc.getRange(2,1,srcLast-1,2).getValues()
.map(r=>({piso:norm(r[0]), url:norm(r[1])}))
.filter(r=>r.piso && r.url);

const bulk = [];
rows.forEach(({piso, url})=>{
try{
  const resp = UrlFetchApp.fetch(url, {muteHttpExceptions:true});
  if (resp.getResponseCode() !== 200) return;
  const text = resp.getContentText();
  const events = parseICS(text);
  events.forEach(ev=>{
    // "Fecha servicio" = día de salida: usamos DTEND; si no hay,
    fallback a DTSTART
    const dtEnd = ev.dtend || ev.dtstart;
    if (!dtEnd) return;

    const fechaSalida = toMidday(new Date(dtEnd));
    if (!inWindow(fechaSalida, from, to)) return;

    const fechaStr = asLocalDateStr(fechaSalida);
}
}
}

```

```

        const tipo = 'Limpieza salida';
        const estado = (String(ev.status||'')).toUpperCase()=='CANCELLED') ?
'Cancelado' : 'Pendiente';
        const uid = norm(ev.uid) ||
`uid_${sha1(` ${piso}|${fechaStr}|${tipo}`)}`;
        const obsIcs = norm(ev.summary || ev.description || '');

        bulk.push([
            fechaStr, // Fecha (salida)
            tipo, // Tipo
            piso, // Piso
            1, // Días (siempre 1)
            estado, // Estado
            keyFrom(piso, fechaStr, tipo), // Clave
            'KrossBooking', // Origen
            uid, // UID (o sintético)
            obsIcs ? `[ICS] ${obsIcs}` : '', // OBS
            asLocalTs(new Date()) // Marca de Tiempo
        ]);
    });
} catch(e){ /* quiet */ }
});

if (bulk.length){
    shTmp.getRange(2,1,bulk.length, header.length).setValues(bulk);
}
}

function parseICS(text){
    const lines = text.replace(/\r/g,'').split('\n');
    const out = []; let ev=null;
    const unfold = [];

```

```

for (let i=0;i<lines.length;i++){
  const l = lines[i];
  if (l.startsWith(' ') || l.startsWith('\t')) unfold[unfold.length-1] += l.trim();
  else unfold.push(l);
}

unfold.forEach(line=>{
  if (line==='BEGIN:VEVENT') { ev={}; return; }
  if (line==='END:VEVENT') { if (ev) out.push(ev); ev=null; return; }
  if (!ev) return;
  const [rawK, ...rest] = line.split(':');
  const v = rest.join(':');
  if (!rawK) return;
  const k = rawK.toUpperCase();
  if (k.startsWith('DTSTART')) ev.dtstart = icsToDate(v);
  else if (k.startsWith('DTEND')) ev.dtend = icsToDate(v);
  else if (k==='UID') ev.uid = v;
  else if (k==='STATUS') ev.status = v;
  else if (k==='SUMMARY') ev.summary = v;
  else if (k==='DESCRIPTION') ev.description = v;
});
return out;
}

function icsToDate(v){
  if (!v) return null;
  const m =
v.match(/^\d{4}(\d{2})(\d{2})(?:T(\d{2})(\d{2})(\d{2})(Z)?$/);
  if (!m) return null;
  const y=+m[1], M=+m[2]-1, d=+m[3];
  if (!m[4]) return toMidday(new Date(y, M, d)); // all-day
  const hh=+m[4], mm=+m[5], ss=+m[6], z=m[7]==='Z';
  return z ? new Date(Date.UTC(y, M, d, hh, mm, ss)) : new Date(y, M, d, hh, mm, ss);
}

```

```

function applyResumenTestToResumen(){
    ensureResumenHeader_();           // ← asegura cabecera
    const shTmp = getSheet(SHEET_RESUMEN_TEST);
    const shDst = getSheet(SHEET_RESUMEN);
    const shBase = getSheet(SHEET_BASE_DATOS);
    if (!shTmp || !shDst) throw new Error('Falta RESUMEN_TEST o Resumen');

    const now = new Date();
    const from = startOfDay(new Date(now));
    const to = endOfDay(new Date(now));
    to.setDate(to.getDate() + IMPORT_FUTURE_DAYS);

    const dstHeader = shDst
        .getRange(1,1,1,Math.max(1,shDst.getLastColumn()))
        .getValues()[0]
        .map(String);

    const idx = (name)=> dstHeader.indexOf(name);
    const iFecha = idx('Fecha'),
          iTipo = idx('Tipo'),
          iPiso = idx('Piso'),
          iDias = idx('Días'),
          iEstado = idx('Estado'),
          iClave = idx('Clave'),
          iOrigen = idx('Origen'),
          iUID = idx('UID'),
          iOBS = idx('OBS'),
          iEventID = idx('EventID'),
          iTS = idx('Marca de Tiempo');

    // Mapa de existentes por Clave
    const existing = new Map();
    const lastDst = shDst.getLastRow();

```

```

if (lastDst >= 2){

    const vals = shDst.getRange(2,1,lastDst-1,
dstHeader.length).getValues();

    vals.forEach((row,r)=>{
        const clave = (iClave >= 0) ? norm(row[iClave]) : '';
        if (clave) existing.set(clave, {row:r+2, data:row});
    });
}

// BASE_DATOS

const baseMap = {};

if (shBase){

    const bh      =
shBase.getRange(1,1,1,shBase.getLastColumn()).getValues()[0].map(String);

    const bLast = shBase.getLastRow();

    if (bLast >= 2){

        const allB   = shBase.getRange(2,1,bLast-1,bh.length).getValues();
        const iBPiso = bh.findIndex(h => String(h).toLowerCase() === 'piso');

        allB.forEach(row=>{
            const piso = norm(row[iBPiso]);
            if (piso) {
                const rec = {};
                bh.forEach((h,j)=> rec[h] = row[j]);
                baseMap[piso] = rec;
            }
        });
    }

    const lastTmp = shTmp.getLastRow();

    if (lastTmp < 2) return;

    const tmpVals = shTmp.getRange(2,1,lastTmp-1,
shTmp.getLastColumn()).getValues();
}

```

```

const toDate = (v)=>{
  if (v instanceof Date && !isNaN(v)) return toMidday(new Date(v));
  const m = String(v||'').match(/^\d{1,2})\/( \d{1,2})\/( \d{4})$/);
  if (m) return toMidday(new Date(+m[3], +m[2]-1, +m[1]));
  const d = new Date(v);
  if (!isNaN(d)) return toMidday(d);
  return null;
};

const appends = [];
const toNotify = [] // índices dentro de appends que habrá que mandar al webhook

tmpVals.forEach(row=>{
  const fechaStr = norm(row[0]);
  const tipo      = norm(row[1]);    // "Limpieza salida"
  const piso      = norm(row[2]);
  const estado    = norm(row[4]);    // Pendiente / Cancelado
  const clave     = norm(row[5]);
  const origen    = norm(row[6]);    // "KrossBooking"
  const uid       = norm(row[7]);
  const obsIcs   = norm(row[8]);

  const fecha = toDate(fechaStr);
  if (!fecha || !inWindow(fecha, from, to) || !piso || !clave) return;

  const base = baseMap[piso] || null;

  // Cancelaciones desde ICS
  if (estado.toUpperCase() === 'CANCELADO'){
    const ex = existing.get(clave);
    if (ex){
      if (iEstado >= 0) shDst.getRange(ex.row,
iEstado+1).setValue('Cancelado');
    }
  }
}

```

```

        if (iTS      >= 0) shDst.getRange(ex.row,
iTS+1).setValue(asLocalTs(new Date()));
    }
    return;
}

const ex = existing.get(clave);
if (ex){
    // 🔒 Modo conservador: solo tocamos si hay cambios reales
    let touched = false;

    // Origen: solo si está vacío
    if (iOrigen >= 0){
        const curOrigen = String(shDst.getRange(ex.row,
iOrigen+1).getValue() || '');
        if (!curOrigen && origen){
            shDst.getRange(ex.row, iOrigen+1).setValue(origen || 'KrossBooking');
            touched = true;
        }
    }

    // UID: solo si está vacío
    if (iUID >= 0 && uid){
        const curUid = String(shDst.getRange(ex.row, iUID+1).getValue() || '');
        if (!curUid){
            shDst.getRange(ex.row, iUID+1).setValue(uid);
            touched = true;
        }
    }

    // OBS: solo añadimos obsIcs si aún no está dentro
    if (iOBS >= 0 && obsIcs){

```

```

        const prev = String(shDst.getRange(ex.row, i0BS+1).getValue() || '');
    }

    if (!prev.includes(obsIcs)){
        shDst.getRange(ex.row, i0BS+1).setValue(prev ?
`$${prev}\n${obsIcs}` : obsIcs);
        touched = true;
    }
}

// Estado: Pendiente solo si no estaba Cancelado ni ya Pendiente
if (iEstado >= 0){
    const curEstado = String(shDst.getRange(ex.row,
iEstado+1).getValue() || '');
    if (!/^Cancelado$/i.test(curEstado) && curEstado !== 'Pendiente'){
        shDst.getRange(ex.row, iEstado+1).setValue('Pendiente');
        touched = true;
    }
}

// Marca de tiempo solo si se ha tocado algo
if (touched && iTS >= 0){
    shDst.getRange(ex.row, iTS+1).setValue(asLocalTs(new Date()));
}

// No pasamos al bloque "Nuevo"
return;
}

// === Nuevo registro desde ICS ===
const payload = new Array(dstHeader.length).fill('');
if (iFecha >= 0) payload[iFecha] = fecha; // ← AQUÍ el cambio: guardamos Date, no string
if (iTipo >= 0) payload[iTipo] = tipo;
if (iPiso >= 0) payload[iPiso] = piso;
if (iDias >= 0) payload[iDias] = 1;

```

```

    if (iEstado >= 0) payload[iEstado] = 'Pendiente';
    if (iClave >= 0) payload[iClave] = clave;
    if (iOrigen >= 0) payload[iOrigen] = origen || 'KrossBooking';
    if (iUID >= 0) payload[iUID] = uid;
    if (iOBS >= 0) payload[iOBS] = obsIcs;

    if (!base){
        if (iEstado >= 0) payload[iEstado] = 'ERROR';
        if (iOBS >= 0){
            payload[iOBS] = (payload[iOBS] ? `${payload[iOBS]}\n` : '') + 'piso
no encontrado en Base_Datos';
        }
    } else {
        // Rellenar columnas desde BASE_DATOS si están vacías
        Object.keys(base).forEach(col=>{
            const pos = dstHeader.indexOf(col);
            if (pos >= 0 && (payload[pos] === '' || payload[pos] == null)){
                payload[pos] = base[col];
            }
        });
    }
}

if (iTS >= 0) payload[iTS] = asLocalTs(new Date());

// Guardamos posición si el Estado queda en Pendiente (solo esos
disparan webhook)
if (iEstado >= 0 && payload[iEstado] === 'Pendiente') {
    toNotify.push(appends.length); // índice que tendrá este payload
}

appends.push(payload);
existing.set(clave, {row: (lastDst + appends.length), data: payload});
});

if (appends.length){

```

```

    const firstRow = Math.max(2, lastDst + 1);
    shDst
        .getRange(firstRow, 1, appends.length, dstHeader.length)
        .setValues(appends);

    // Disparar webhook SOLO para las filas nuevas "Pendiente"
    toNotify.forEach(idx => {
        const rowNum = firstRow + idx;
        try {
            notificarWebhook(rowNum);
        } catch (e) {
            Logger.log(`Error enviando ICS-Resumen fila ${rowNum} al webhook:
${e}`);
        }
    });
}
}

```

```

/*
=====
=====
4) WATCHER NOCTURNO 03:05 - Calendar → Resumen (relee TODO el rango 30
días)
    - Título cambiado en Calendar → añadir a OBS (versionado) y
Estado=Modificado
    - Fecha cambiada → actualizar Fecha + Clave y Estado=Modificado
    - Borrado/Cancelado en Calendar → Estado=Cancelado (conserva EventID
limpio si procede)
=====
===== */

```

```

function syncCalendarChangesToResumen(){
  const sh = getSheet(SHEET_RESUMEN);
  if (!sh) { Logger.log('✖ Falta hoja Resumen'); return; }

  const H =
    sh.getRange(1,1,1,sh.getLastColumn()).getValues()[0].map(String);
  const iFecha = H.indexOf('Fecha'), iPiso=H.indexOf('Piso'),
  iTipo=H.indexOf('Tipo'),
      iEstado=H.indexOf('Estado'), iClave=H.indexOf('Clave'),
  iOBS=H.indexOf('OBS'),
      iEventID=H.indexOf('EventID'), iTS=H.indexOf('Marca de Tiempo');

  const last = sh.getLastRow();
  if (last < 2) return;

  const now = new Date();
  const from = startOfDay(new Date(now));
  const to = endOfDay(new Date(now));
  to.setDate(to.getDate()+IMPORT_FUTURE_DAYS);

  // Indexar filas de Resumen en ventana por EventID
  const data = sh.getRange(2,1,last-1,H.length).getValues();
  const byId = new Map();
  data.forEach((row, rIdx)=>{
    const fecha = row[iFecha];
    const d = (fecha instanceof Date && !isNaN(fecha)) ? toMidday(new Date(fecha)) : null;
    if (!d || !inWindow(d, from, to)) return;
    const evId = norm(row[iEventID]);

```

```
    if (evId) byId.set(evId, {row: rIdx+2, data: row});
});

// Releer TODO el rango 30 días por cada calendario (sin updatedMin)
let mod=0, canc=0, nohit=0;
for (const cliente in CALENDARIOS){
  const calId = CALENDARIOS[cliente];

  let pageToken = null;
  do{
    const resp = Calendar.Events.list(calId, {
      timeMin: from.toISOString(),
      timeMax: addDays(to,1).toISOString(),
      singleEvents: true,
      showDeleted: true,
      maxResults: 2500,
      pageToken
    });
    const items = (resp && resp.items) || [];
    for (const e of items){
      const id      = e.id;
      const title   = e.summary || '';
      const deleted = (e.status === 'cancelled');

      // Buscar por EventID exacto (regla confirmada)
      const hit = byId.get(id);
      if (!hit){ nohit++; continue; }

      const row = hit.row;
```

```

    if (deleted){
        if (iEstado>=0) sh.getRange(row, iEstado+1).setValue('Cancelado');
        if (iTS>=0)     sh.getRange(row, iTS+1).setValue(asLocalTs(new Date()));
    }
}

let touched = false;

// Título cambiado: anotar en OBS y marcar Modificado
const pisoActual = norm(sh.getRange(row, iPiso+1).getValue());
if (title && title !== pisoActual){
    if (iOBS>=0){
        const prev = String(sh.getRange(row, iOBS+1).getValue() || '');
        sh.getRange(row, iOBS+1).setValue(prev ? `${prev}\n[CAL] Título
modificado: ${title}` : `[CAL] Título modificado: ${title}`);
    }
    touched = true;
}

// Fecha (all-day): usamos END - 1 como día de salida
const startRaw = e.start && (e.start.date || e.start.dateTime);
const endRaw   = e.end   && (e.end.date   || e.end.dateTime);
if (startRaw && endRaw){
    const end = new Date(endRaw);
    const salida = toMidday(new Date(end));
    salida.setDate(salida.getDate()-1);
    const fechaStrCal = asLocalDateStr(salida);
    const fechaResumen = sh.getRange(row, iFecha+1).getValue();
}

```

```

        const fechaResumenStr = asLocalDateStr(toMidday(new
Date(fechaResumen)));
        if (fechaStrCal !== fechaResumenStr){
            if (iFecha>=0) sh.getRange(row, iFecha+1).setValue(fechaStrCal);
            if (iClave>=0){
                const tipo = norm(sh.getRange(row, iTipo+1).getValue()) ||
'Limpieza salida';
                const piso = norm(sh.getRange(row, iPiso+1).getValue());
                sh.getRange(row, iClave+1).setValue(keyFrom(piso, fechaStrCal,
tipo));
            }
            touched = true;
        }
    }

    if (touched){
        if (iEstado>=0) sh.getRange(row,
iEstado+1).setValue('Modificado');
        if (iTS>=0)     sh.getRange(row, iTS+1).setValue(asLocalTs(new
Date()));
        mod++;
    }
}

pageToken = resp.nextPageToken;
} while(pageToken);
}

Logger.log(`Watcher (30d) → Modificados:${mod}  Borrados:${canc}
SinCoincidencia:${nohit}`);
}

```

```

/** Trigger diario recomendado (03:05) */
function createTrigger_syncCalendarDaily(){

ScriptApp.newTrigger('syncCalendarChangesToResumen').timeBased().atHour(3).
nearMinute(5).everyDays(1).create();
}

/*
=====
=====
5) PUSH Resumen → Calendarios (crea/actualiza all-day) – sin "guard
Calendario"
===== * /

```



```

function findExistingEventId(calId, dateObj, piso){
  const timeMin = new Date(dateObj.getFullYear(), dateObj.getMonth(),
dateObj.getDate()).toISOString();
  const timeMax = new Date(dateObj.getFullYear(), dateObj.getMonth(),
dateObj.getDate()+1).toISOString();

  const res = Calendar.Events.list(calId, { timeMin, timeMax,
singleEvents:true, showDeleted:false, maxResults: 50, orderBy:'startTime'
});
  const target = _normTxt(piso);
  const items = (res && res.items) || [];
  for (const ev of items){
    const title = _normTxt(ev.summary || '');

```

```

        if (title.includes(target) || target.includes(title)) return ev.id || null;
    }
    return null;
}

function deleteEventSafe(calId, eventId){
    try{ Calendar.Events.remove(calId, eventId); return true; }
    catch(e){ Logger.log('⚠️ No pude borrar '+eventId+' de '+calId+': '+e);
    return false; }
}

/***
 * LEGACY wrapper:
 * Antes empujaba directamente a Google Calendar.
 * Ahora solo reenvía al webhook todas las filas Pendiente sin EventID,
 * delegando en pushResumenToCalendars_viaWebhook().
*/
function pushResumenToCalendars() {
    return pushResumenToCalendars_viaWebhook();
}

/*
=====
=====
6) UTILIDADES HOJA
=====
=====
*/
function quitarDominioID() {
    const hoja = getSheet(SHEET_RESUMEN);
    const ultima = hoja.getLastRow();
}

```

```

if (ultima < 2) return;

const rango = hoja.getRange(2, 10, ultima - 1, 1); // J2:J
const valores = rango.getValues();
const sinDominio = valores.map(fila => {
  const id = fila[0];
  return [ id && id.toString().includes("@") ? id.toString().split("@")[0]
: id ];
});
rango.setValues(sinDominio);
}

/*
=====
=====

7) KAWAKAN – ahora también vía Webhook/Make
=====

=====
===== */

const KAWA = {
  CLIENTE: 'KAWAKAN SL',
  PISO: 'Av. Diagonal 600',
  DIAS_SEMANA: [1,3,5], // L M X J V → 1,3,5 = L,M,X
  HOJA_RESUMEN: SHEET_RESUMEN,
  // Si es true: al crear las filas nuevas en Resumen, se envían
  // automáticamente al webhook
  PUBLICAR_ALCALENDARIO_POR_DEFECTO: true,
  ORIGEN_TAG: 'AUTO_KAWAKAN'
};

function kawa_mesSiguiente_(baseDate = new Date()){
  const y = baseDate.getFullYear(), m = baseDate.getMonth();

```

```

return (m === 11) ? { yy: y+1, mm: 1 } : { yy: y, mm: m+2 }; // 1..12
}

/***
* Genera servicios KAWAKAN en Resumen (mes yyyy/mm).
* - No duplica días ya existentes para ese cliente+piso.
* - Deja las filas nuevas en Estado = "Pendiente".
* - Si publish = true (o nulo y PUBLICAR_ALCALENDARIO_POR_DEFECTO = true):
*     dispara notificarWebhook(row) para cada fila nueva Pendiente.
*/
function generarServiciosKawakan_Run(yyyy, mm, publish){
  const sh = getSheet(KAWA.HOJA_RESUMEN);
  const head =
    sh.getRange(1,1,1,sh.getLastColumn()).getDisplayValues()[0].map(x =>
      String(x||'').trim());
  const normH = s =>
    s.normalize('NFD').replace(/[\u0300-\u036f]/g,'').toLowerCase().replace(/\s+/g,'');
  const H = {}; head.forEach((h,i)=> H[normH(h)] = i);
  const need = (name, alts=[]) => {
    const cands = [name, ...alts].map(normH);
    for (const c of cands){
      if (H[c] != null) return H[c];
    }
    return null;
  };

  const idxCliente = need('cliente',['cliente(origen)']);
  const idxPiso    = need('piso');
  const idxFecha   = need('fecha',['inicio']);
  const idxEstado  = need('estado',['estado(origen)']);
  const idxOrigen  = need('origen');

  if (idxCliente==null || idxPiso==null || idxFecha==null)

```

```

throw new Error('Resumen necesita columnas Cliente, Piso y Fecha.');

const last = sh.getLastRow();
const existentes = new Set();
if (last>=2){
    const vals = sh.getRange(2,1,last-1,sh.getLastColumn()).getValues();
    const fmt = d => ymd(d instanceof Date ? d : new Date(String(d||'')));
    for (const r of vals){
        const cli = String(r[idxCliente]||'').trim();
        const piso= String(r[idxPiso]||'').trim();
        const fec = r[idxFecha];
        if (cli && piso && fec) existentes.add(`${cli}|${piso}|${fmt(fec)}`);
    }
}

const nuevos = [];
const rowTemplate = new Array(sh.getLastColumn()).fill('');

for (let d = 1; d <= 31; d++){
    const day = new Date(yyyy, mm-1, d);
    if (day.getMonth() !== (mm-1)) break; // fin de mes
    if (!KAWA.DIAS_SEMANA.includes(day.getDay())) continue; // solo L/M/X
según config

    const key = `${KAWA.CLIENTE}|${KAWA.PISO}|${ymd(day)}`;
    if (existentes.has(key)) continue; // ya existe, no duplicar

    const row = rowTemplate.slice();
    row[idxCliente] = KAWA.CLIENTE;
    row[idxPiso]     = KAWA.PISO;
    row[idxFecha]   = day;

    // Estado inicial → Pendiente (lo gestionará Make)
    if (idxEstado != null) row[idxEstado] = 'Pendiente';
}

```

```

// Origen para rastrear que vienen de KAWAKAN
if (idxOrigen != null) row[idxOrigen] = KAWA.ORIGEN_TAG;

nuevos.push(row);
existentes.add(key);
}

if (!nuevos.length){
  SpreadsheetApp.getActive().toast('KAWAKAN: sin nuevas filas
(idempotente).', 'Operaciones', 5);
  return 0;
}

const startRow = sh.getLastRow() + 1;
sh.insertRowsAfter(sh.getLastRow(), nuevos.length);
sh.getRange(startRow, 1, nuevos.length,
sh.getLastColumn()).setValues(nuevos);

// ¿Debemos enviarlos directamente a Make?
const shouldPublish = (publish === true) || (publish == null &&
KAWA.PUBLICAR_AL_CALENDARIO POR_DEFECTO);

if (shouldPublish){
  for (let i = 0; i < nuevos.length; i++){
    const rowNum = startRow + i;
    try {
      notificarWebhook(rowNum);
    } catch (e) {
      Logger.log(`KAWAKAN: error enviando fila ${rowNum} al webhook:
${e}`);
    }
  }
}

SpreadsheetApp.getActive().toast(

```

```

`KAWAKAN: añadidas ${nuevos.length} filas para
${Utilities.formatDate(new Date(yyyy, mm-1, 1), TZ, 'MMMM yyyy')}` ,
'Operaciones',
6
);
return nuevos.length;
}

function generarServiciosKawakanNow(){
const {yy, mm} = kawa_mesSiguiente_();
return generarServiciosKawakan_Run(yy, mm, null);
}

function generarServiciosKawakan_Auto(){
const today = new Date();
if (today.getDate() !== 28) return; // solo día 28
const {yy, mm} = kawa_mesSiguiente_(today);
generarServiciosKawakan_Run(yy, mm, true); // auto publicar vía webhook
}

function createMonthlyTrigger_Kawakan(){
ScriptApp.getProjectTriggers().forEach(t=>{
const h = t.getHandlerFunction ? t.getHandlerFunction() : '';
if (h==='generarServiciosKawakan_Auto') ScriptApp.deleteTrigger(t);
});
ScriptApp.newTrigger('generarServiciosKawakan_Auto')
.timeBased()
.onMonthDay(28)
.atHour(9)
.create();
SpreadsheetApp.getActive().toast('Trigger KAWAKAN creado (día 28 ·
09:00).', 'Operaciones', 6);
}

```

```
/*
=====
=====

 8) ORQUESTADOR + TRIGGERS

=====
===== */

function runIcsEvery2h(){
  syncIcsSnapshotToResumenTest(); // Snapshot limpio 30 días → RESUMEN_TEST
  applyResumenTestToResumen();    // Aplica a Resumen (ICS prioriza)
}

function installTriggers(){
  // Limpia todos los triggers actuales
  ScriptApp.getProjectTriggers().forEach(t => ScriptApp.deleteTrigger(t));

  // ICS cada 2h EN PUNTO
  ScriptApp.newTrigger('runIcsEvery2h')
    .timeBased()
    .everyHours(2)
    .atHour(0)
    .create();

  // Watcher 03:05
  ScriptApp.newTrigger('syncCalendarChangesToResumen')
    .timeBased()
    .atHour(3)
    .nearMinute(5)
    .everyDays(1)
    .create();

  // Vuelve a crear también el de Form Submit
}
```

```
installFormSubmitTrigger();

SpreadsheetApp.getActive().toast(
  'Triggers instalados: ICS cada 2h, Watcher 03:05 y Form Submit.',
  'Operaciones',
  6
);
return 'Triggers instalados';
}

/**
* Crea el trigger instalable de Form Submit → llama a onFormSubmit(e)
*/
function installFormSubmitTrigger() {
  const ss = SpreadsheetApp.getActive();

  // Limpia triggers antiguos de onFormSubmit para no duplicar
  ScriptApp.getProjectTriggers().forEach(t => {
    const h = t.getHandlerFunction && t.getHandlerFunction();
    if (h === 'onFormSubmit') {
      ScriptApp.deleteTrigger(t);
    }
  });

  // Crea el trigger de envío de formulario
  ScriptApp.newTrigger('onFormSubmit')
    .forSpreadsheet(ss)
    .onFormSubmit()
    .create();

  SpreadsheetApp.getActive().toast(
```

```

    'Trigger de Form Submit instalado correctamente.' ,
    'Operaciones' ,
    5
);
}

function ensureResumenHeader_(){
const sh = SpreadsheetApp.getActive().getSheetByName(SHEET_RESUMEN);
if (!sh) return;
const needed =
['Fecha','Tipo','Piso','Días','Estado','Clave','Origen','UID','OBS','EventID','Marca de Tiempo'];
const lastCol = Math.max(1, sh.getLastColumn());
const header = sh.getRange(1,1,1,lastCol).getValues()[0].map(v =>
String(v||''));
const have = new Set(header.filter(Boolean));
let col = header.length;
needed.forEach(n=>{
  if (!have.has(n)){ sh.getRange(1, ++col).setValue(n); }
});

// Forzar formato de fecha en toda la columna "Fecha"
fixFormatoFechaResumen_();
}

/**
 * Reenvía al webhook Make todas las filas de RESUMEN
 * que estén en Estado = "Pendiente" y sin EventID.
 *
 * Útil como botón manual de rescate cuando hay algo

```

```
* que no ha llegado a Make o quieres reempujar pendientes.  
*/  
/**  
 * Reenvía al webhook Make todas las filas de RESUMEN  
 * que estén en Estado = "Pendiente" y sin EventID.  
 *  
 * Ahora en "modo troceado": si se acerca al límite de tiempo  
 * para, muestra un aviso, y puedes volver a ejecutar para seguir.  
 */  
  
function pushResumenToCalendars_viaWebhook() {  
  const sh = getSheet(SHEET_RESUMEN);  
  if (!sh) {  
    Logger.log('✖ Falta hoja Resumen');  
    return;  
  }  
  
  const last = sh.getLastRow();  
  if (last < 2) {  
    SpreadsheetApp.getActive().toast(  
      'No hay filas en Resumen.',  
      'Operaciones',  
      4  
    );  
    return;  
  }  
  
  const header = sh  
    .getRange(1, 1, 1, sh.getLastColumn())  
    .getValues()[0]  
    .map(v => String(v || ''));  
  
  const iEstado = header.indexOf('Estado');  
  const iEventID = header.indexOf('EventID');  
  
  if (iEstado === -1 || iEventID === -1) {
```

```
SpreadsheetApp.getActive().toast(
  'Faltan columnas "Estado" o "EventID" en Resumen.',
  'Operaciones',
  6
);
return;
}

// Leemos TODAS las filas (2..last) de una vez
const data = sh.getRange(2, 1, last - 1, sh.getLastColumn()).getValues();

// Contamos cuántas pendientes sin EventID hay en total (para info)
let pendientesTotal = 0;
data.forEach(row => {
  const estado = String(row[iEstado] || '').toLowerCase();
  const eventId = String(row[iEventID] || '').trim();
  if (estado === 'pendiente' && !eventId) {
    pendientesTotal++;
  }
});

if (pendientesTotal === 0) {
  SpreadsheetApp.getActive().toast(
    'Webhook: no hay filas "Pendiente" sin EventID que reenviar.',
    'Operaciones',
    5
);
return;
}

const startTs = Date.now();
const MAX_MS = (5 * 60 * 1000) - 15000; // ~4:45 min para no llegar al límite

let reenviadas = 0;
```

```

for (let i = 0; i < data.length; i++) {
    const rowData = data[i];
    const estado = String(rowData[iEstado] || '').toLowerCase();
    const eventId = String(rowData[iEventID] || '').trim();
    const rowNum = i + 2; // porque data[0] corresponde a la fila 2

    // Solo reenviamos:
    // - Estado = "pendiente"
    // - Sin EventID (aún no creado/enlazado)
    if (estado === 'pendiente' && !eventId) {
        try {
            notificarWebhook(rowNum);
            reenviadas++;
        } catch (e) {
            Logger.log(`Error reenviando fila ${rowNum} al webhook: ${e}`);
        }
    }

    // Corte de seguridad por tiempo
    if (Date.now() - startTs > MAX_MS) {
        SpreadsheetApp.getActive().toast(
            `Webhook: reenviadas ${reenviadas} de ~${pendientesTotal} pendientes
(tiempo límite). Vuelve a ejecutar para seguir.`,
            'Operaciones',
            8
        );
        Logger.log(
            `Webhook parcial: reenviadas ${reenviadas} de ~${pendientesTotal}
(tiempo límite alcanzado).`);
    }
}

return;
}

```

```
SpreadsheetApp.getActive().toast(  
  `Webhook: reenviadas ${reenviadas} filas pendientes sin EventID.`,
  'Operaciones',
  6
);
Logger.log(
  `Webhook completo: reenviadas ${reenviadas} de ${pendientesTotal}
pendientes.`);
}
```