```
/**
 * 04_Facturación — v2.1 (solo formato <Mes>_Cierre_<YYYY-MM-DD>)
 * - Sin webhook, sin helpers antiguos
 * - Lee "Cierre_mes" del cierre mensual
 */

/***** === CONFIG FACTURACIÓN === *****/
const FACT_CFG = {
  // Carpeta con los cierres: 00_RESUMEN DATOS
  SNAPSHOTS_FOLDER_ID: '1fvcLjNv2-VTyzN5WRaWyPODndHAHuovV',

  // Carpeta destino para archivos que verá Make: 00_RESUMEN CLIENTES
  PENDIENTES_FOLDER_ID: '14QZy5qJc74AzBhEF_lYrJ06PIrXuLuOr',

  // Impuestos, exclusiones y mapping básicos
  IVA_PCT         : 21,
  EXCLUIR_ESTADOS : ['Cancelado','Eliminado'] // por seguridad
};

// Hoja maestra con datos de cliente (en este mismo libro maestro activo)
const FACT_MASTER = {
  HOJA_BASE_DATOS : 'BASE_DATOS',
  COL_BD_CLIENTE  : 'Cliente',
  COL_BD_NIF      : 'NIF',
  COL_BD_DIR      : 'Direccion_Facturacion'
};

/***** === HELPERS NAMESPACE === *****/
const FAC = {
  tz: () => Session.getScriptTimeZone() || 'Europe/Madrid',
  pad2: n => (n < 10 ? '0' + n : '' + n),
  monthName: m0 =>
['Enero','Febrero','Marzo','Abril','Mayo','Junio','Julio','Agosto','Septiem
bre','Octubre','Noviembre','Diciembre'][m0],
  sanitize: s => String(s || '')
```

```
    .normalize('NFD').replace(/[\u0300-\u036f]/g, '')
    .replace(/[^a-zA-Z0-9 _.-]+/g, '').trim(),

  ensureSubfolder: (parent, name) => {
    const it = parent.getFoldersByName(name);
    return it.hasNext() ? it.next() : parent.createFolder(name);
  },

  getFolderByIdSafe: (id) => {
    try { return DriveApp.getFolderById(id); } catch(e){ return null; }
  },

  /**
   * Busca el archivo de cierre del mes: <Mes>_Cierre_<YYYY-MM-DD>
   * Selecciona por YYYY-MM (needle: "_Cierre_YYYY-MM-") y elige el más
reciente.
   */
  findCierreByYYYYMM: (yyyyMM) => {
    const folder = FAC.getFolderByIdSafe(FACT_CFG.SNAPSHOTS_FOLDER_ID);
    if (!folder) throw new Error('No encuentro la carpeta 00_RESUMEN DATOS
(SNAPSHOTS_FOLDER_ID).');

    const needle = `_Cierre_${yyyyMM}-`; // p.ej., _Cierre_2025-11-
    let chosen = null, chosenTs = 0;

    const consider = (file) => {
      if (file.getMimeType() !== MimeType.GOOGLE_SHEETS) return;
      const name = file.getName();
      if (name.includes(needle)) {
        const ts = file.getLastUpdated().getTime();
        if (!chosen || ts > chosenTs){ chosen = file; chosenTs = ts; }
      }
    };

    const it = folder.getFiles();
```

```
    while (it.hasNext()) consider(it.next());

    return chosen; // puede ser null si no hay cierre
  }
}; // <-- cierre limpio de FAC

/***** === MENÚ FACTURACIÓN === *****/
function addMenuFacturacion(){
  SpreadsheetApp.getUi()
    .createMenu('💼 Facturación')
    .addItem('Generar archivo (Cliente + Mes)…',
'solicitarFactura_DriveWatch')
    .addToUi(); // <-- corregido (un solo punto)
}

/***** === UI: selector Cliente + Mes (reutilizable) === *****/
function solicitarFactura_DriveWatch(){
  const hoy = new Date();
  const yDef = hoy.getFullYear();
  const mDef = hoy.getMonth()+1;

  const clientes = getClientesDesdeBaseDatos_();
  if (!clientes.length){
    SpreadsheetApp.getUi().alert('No se han encontrado clientes en
BASE_DATOS!A:A');
    return;
  }
  const html = buildClienteDialog_(clientes, yDef, mDef,
/*handler*/'continuarSolicitudFactura_DriveWatch');
  SpreadsheetApp.getUi().showModalDialog(html, 'Generar archivo para
Make');
}

function getClientesDesdeBaseDatos_(){
  const ss = SpreadsheetApp.getActive();
```

```
  const sh = ss.getSheetByName(FACT_MASTER.HOJA_BASE_DATOS);
  if (!sh) throw new Error('No encuentro la hoja BASE_DATOS');
  const lastRow = sh.getLastRow();
  if (lastRow < 2) return [];
  const vals = sh.getRange(2,1,lastRow-1,1).getDisplayValues()
    .map(r => String(r[0]||'').trim())
    .filter(v => v);
  const norm = s =>
s.normalize('NFD').replace(/[\u0300-\u036f]/g,'').toLowerCase();
  const seen = new Set(), uniques = [];
  for (const v of vals){ const k=norm(v); if(!seen.has(k)){ seen.add(k);
uniques.push(v);} }
  uniques.sort((a,b)=> a.localeCompare(b,'es',{sensitivity:'base'}));
  return uniques;
}

function buildClienteDialog_(clientes, yDef, mDef, handlerFn){
  const months = Array.from({length:12},(_,i)=>{
    const m = (i+1).toString().padStart(2,'0');
    const sel = (i+1)===mDef ? ' selected' : '';
    return `<option value="${m}"${sel}>${FAC.monthName(i)}
(${m})</option>`;
  }).join('');
  const years = Array.from({length:5},(_,k)=>{
    const y = (yDef - 2 + k);
    const sel = y===yDef ? ' selected' : '';
    return `<option value="${y}"${sel}>${y}</option>`;
  }).join('');
  const optionsClientes = clientes.map(c=>`<option
value="${c}">${c}</option>`).join('');

  const html = HtmlService.createHtmlOutput(`
    <div style="font:14px system-ui,Segoe UI,Roboto,Arial; padding:14px
16px;">
```

```
      <h2 style="margin:0 0 10px; font-size:16px;">Generar archivo (Cliente
+ Mes)</h2>
      <div style="display:grid; grid-template-columns: 1fr 1fr; gap:12px;">
        <div>
          <label style="display:block; margin:0 0 4px;">Mes</label>
          <select id="mes" style="width:100%;
padding:6px;">${months}</select>
        </div>
        <div>
          <label style="display:block; margin:0 0 4px;">Año</label>
          <select id="anio" style="width:100%;
padding:6px;">${years}</select>
        </div>
      </div>
      <label style="display:block; margin:12px 0 4px;">Cliente</label>
      <select id="cliente" style="width:100%;
padding:6px;">${optionsClientes}</select>
      <div id="msg" style="margin-top:10px; color:#555;"></div>
      <div style="margin-top:16px; display:flex; gap:8px;
justify-content:flex-end;">
        <button type="button" onclick="google.script.host.close()"
style="padding:6px 12px;">Cancelar</button>
        <button id="go" type="button" onclick="enviar()" style="padding:6px
12px;">Generar</button>
      </div>
      <script>
        function enviar(){
          var btn = document.getElementById('go');
          var msg = document.getElementById('msg');
          btn.disabled = true; msg.textContent = 'Procesando…';
          var cliente = document.getElementById('cliente').value;
          var mm      = document.getElementById('mes').value;
          var yy      = document.getElementById('anio').value;
          var ym      = yy + '-' + mm; // YYYY-MM
```

```
          if(!cliente){ alert('Selecciona un cliente'); btn.disabled=false;
msg.textContent=''; return; }
          google.script.run
            .withSuccessHandler(function(){ google.script.host.close(); })
            .withFailureHandler(function(err){ btn.disabled=false;
msg.textContent=''; alert(err && err.message ? err.message : err); })
            [${JSON.stringify(handlerFn)}]({ cliente: cliente, yyyyMM: ym
});
        }
      </script>
    </div>
  `);
  html.setWidth(560).setHeight(320);
  return html;
}

/***** === BACKEND NUEVO (Drive Watch) === *****/
function continuarSolicitudFactura_DriveWatch(data){
  const yyyyMM = String(data?.yyyyMM || '').trim();
  const cliente = String(data?.cliente || '').trim();
  if (!/^\d{4}-\d{2}$/.test(yyyyMM)) throw new Error('Mes inválido
(esperado YYYY-MM).');
  if (!cliente) throw new Error('Cliente vacío.');
  if (!FACT_CFG.PENDIENTES_FOLDER_ID) throw new Error('Configura
FACT_CFG.PENDIENTES_FOLDER_ID');

  // 1) Localizar archivo de CIERRE del mes
  const snap = FAC.findCierreByYYYYMM(yyyyMM);
  if (!snap) throw new Error(`No encuentro archivo de cierre para
"${yyyyMM}" (buscado patrón "_Cierre_${yyyyMM}-" en 00_RESUMEN DATOS).`);

  // 2) Abrir y leer la hoja Cierre_mes
  const ssSnap = SpreadsheetApp.openById(snap.getId());
  const sh = ssSnap.getSheetByName('Cierre_mes');
```

```
  if (!sh) throw new Error(`En ${snap.getName()} no existe la hoja
Cierre_mes`);

  const lastRow = sh.getLastRow(), lastCol = sh.getLastColumn();
  const vals = sh.getRange(1,1,lastRow,lastCol).getValues();
  const head = vals.shift();

  // 3) Índices robustos (acepta variantes y diacríticos)
  const norm = s =>
String(s||'').normalize('NFD').replace(/[\u0300-\u036f]/g,'').toLowerCase()
.trim();
  const H = {}; head.forEach((h,i)=> H[norm(h)] = i);

  const pickIdx = (...names)=>{
    for (const n of names){ const i = H[norm(n)]; if (i != null) return i;
}
    return null;
  };

  const idxCliente      = pickIdx('Cliente (Origen)','Cliente');
  const idxFecha        = pickIdx('Inicio','Fecha'); // en cierre: Inicio
(Date)
  const idxPiso         = pickIdx('Piso');
  const idxEstado       = pickIdx('Estado (Origen)','Estado');
  const idxPL_enriq     = pickIdx('Precio Limpieza');
  const idxPL_origen    = pickIdx('Precio Limpieza (Origen)');
  const idxPR_enriq     = pickIdx('Precio Ropa');
  const idxPR_origen    = pickIdx('Precio Ropa (Origen)');
  const idxTotal_enriq  = pickIdx('Total');
  const idxTotal_origen = pickIdx('Total (Origen)');

  if (idxCliente == null || idxFecha == null || idxPiso == null){
    throw new Error('Faltan columnas básicas (Cliente/Inicio/Piso) en
Cierre_mes.');
  }
```

```
  const tz    = FAC.tz();
  const excl  = FACT_CFG.EXCLUIR_ESTADOS || [];
  const parseFecha = v => v instanceof Date ? v : new Date(String(v||''));
  const yymm = d => Utilities.formatDate(d, tz, 'yyyy-MM');
  const fmtDMY = d => Utilities.formatDate(d, tz, 'dd/MM/yyyy');
  const toNum = v => (typeof v==='number')? v :
(parseFloat(String(v||'').replace(',','.'))||0);
  const r2   = n => Math.round((+n||0)*100)/100;

  // 4) Filtrar por cliente + mes y excluir estados si aplica
  const filas = vals
    .filter(r => String(r[idxCliente]||'').trim() === cliente)
    .filter(r => { const est = idxEstado!=null ?
String(r[idxEstado]||'').trim() : ''; return !excl.includes(est); })
    .filter(r => yymm(parseFecha(r[idxFecha])) === yyyyMM)
    .sort((a,b)=> parseFecha(a[idxFecha]) - parseFecha(b[idxFecha]));

  if (!filas.length) throw new Error('No hay filas para ese cliente en ese
mes/año en el cierre.');

  // 5) Preparar LINEAS: prioriza precios enriquecidos; cae a (Origen)
  const dataLineas = filas.map(r => {
    const d  = parseFecha(r[idxFecha]);
    const pl = (idxPL_enriq!=null ? toNum(r[idxPL_enriq]) : 0) ||
(idxPL_origen!=null ? toNum(r[idxPL_origen]) : 0);
    const pr = (idxPR_enriq!=null ? toNum(r[idxPR_enriq]) : 0) ||
(idxPR_origen!=null ? toNum(r[idxPR_origen]) : 0);
    const tot = (idxTotal_enriq!=null ? toNum(r[idxTotal_enriq]) : 0) ||
(idxTotal_origen!=null ? toNum(r[idxTotal_origen]) : (r2(pl+pr)));
    return [ fmtDMY(d), r[idxPiso], pl, pr, r2(tot) ];
  });

  const subtotal = r2(dataLineas.reduce((s,row)=> s + (row[4]||0), 0));
  const iva      = r2(subtotal * (FACT_CFG.IVA_PCT/100));
```

```
  const total    = r2(subtotal + iva);

  // 6) Crear archivo nuevo en 00_RESUMEN CLIENTES con LINEAS, RESUMEN y
META
  const carpeta = DriveApp.getFolderById(FACT_CFG.PENDIENTES_FOLDER_ID);
  const [yy, mm] = yyyyMM.split('-').map(x=>parseInt(x,10));
  const tzNow = FAC.tz();
  const stamp = Utilities.formatDate(new Date(), tzNow, 'yyyyMMdd_HHmmss');
  const nombreLimpio = cliente.replace(/[^\w]+/g,'_');
  const newName = `FACT_${yyyyMM}_${nombreLimpio}_${stamp}`;

  const nuevo = SpreadsheetApp.create(newName);
  DriveApp.getFileById(nuevo.getId()).moveTo(carpeta);
  const ssNuevo = SpreadsheetApp.openById(nuevo.getId());

  // Hoja LINEAS
  const shLineas = ssNuevo.getSheets()[0];
  shLineas.setName('LINEAS');
  const cab = ['Fecha','Piso','Precio Limpieza','Precio Ropa','Total'];
  shLineas.getRange(1,1,1,cab.length).setValues([cab]);

shLineas.getRange(2,1,dataLineas.length,cab.length).setValues(dataLineas);
  shLineas.autoResizeColumns(1, cab.length);
  shLineas.setFrozenRows(1);

  // RESUMEN agrupado por Piso
  const byPiso = new Map();
  const mode = arr => { if(!arr.length) return 0; const c=new Map();
for(const v of arr){c.set(v,(c.get(v)||0)+1);} return
[...c.entries()].sort((a,b)=> b[1]-a[1] || b[0]-a[0])[0][0]; };
  for (const [fechaStr, piso, pl, pr, tot] of dataLineas){
    const dia = fechaStr ? String(fechaStr).slice(0,2) : '';
    const key = String(piso||'');
    const plN = +pl || 0, prN = +pr || 0, totN = +tot || 0;
```

```
    if (!byPiso.has(key)) byPiso.set(key, { dias: [], plUnits: [], prUnits:
[], total: 0, count: 0 });
    const o = byPiso.get(key);
    if (dia) o.dias.push(dia);
    if (plN>0) o.plUnits.push(r2(plN));
    if (prN>0) o.prUnits.push(r2(prN));
    o.total = r2(o.total + totN);
    o.count++;
  }
  const shResumen = ssNuevo.insertSheet('RESUMEN');
  const cabRes = ['Piso','Dias','Total dias','Precio Limpieza','Precio
Ropa','Total'];
  const out = [];
  for (const [piso, o] of byPiso.entries()){
    const diasUnicos = [...new Set(o.dias)].sort((a,b)=>
parseInt(a,10)-parseInt(b,10));
    const diasText = diasUnicos.join('-');
    the_pl = o.plUnits.length ? mode(o.plUnits) : 0;
    const plUnit = o.plUnits.length ? mode(o.plUnits) : 0;
    const prUnit = o.prUnits.length ? mode(o.prUnits) : 0;
    out.push([piso, diasText, o.count, plUnit, prUnit, r2(o.total)]);
  }
  if (out.length){

shResumen.getRange(1,1,1,cabRes.length).setValues([cabRes]).setFontWeight('
bold');
    shResumen.getRange(2,1,out.length,cabRes.length).setValues(out);
    shResumen.setFrozenRows(1);
    shResumen.autoResizeColumns(1, cabRes.length);
    shResumen.getRange(2,4,out.length,3).setNumberFormat('0.00');
    shResumen.getRange(2,2,out.length,1).setNumberFormat('@');
  }

  // META
  const shMeta = ssNuevo.insertSheet('META');
```

```
  const mesTexto = `${FAC.monthName(mm-1)} ${yy}`;
  const meta = [
    ['cliente', cliente],
    ['mes_key', yyyyMM],
    ['mes_texto', mesTexto],
    ['fecha_emision', Utilities.formatDate(new Date(), tzNow,
'dd/MM/yyyy')],
    ['iva_pct', FACT_CFG.IVA_PCT],
    ['subtotal', subtotal],
    ['iva', iva],
    ['total', total]
  ];

  // NIF y dirección desde BASE_DATOS del libro maestro
  const ssMaster = SpreadsheetApp.getActive();
  const shBD = ssMaster.getSheetByName(FACT_MASTER.HOJA_BASE_DATOS);
  let nifVal = '', dirVal = '';
  if (shBD){
    const bd = shBD.getDataRange().getDisplayValues();
    const h  = bd.shift();
    const m  = {}; const nn = x=>
String(x||'').normalize('NFD').replace(/[\u0300-\u036f]/g,'').toLowerCase()
.replace(/[\s_\/\.-]+/g,'');
    h.forEach((x,i)=> m[nn(x)] = i);
    const idxClienteBD = m[nn(FACT_MASTER.COL_BD_CLIENTE)] ?? m['cliente'];
    const aliasNif =
['nif','cif','nifcif','cifnif','dni','nie','idfiscal','identificacionfiscal
','identificaciónfiscal','numerofiscal','numfiscal','docfiscal','documentof
iscal'];
    let idxNif = m[nn(FACT_MASTER.COL_BD_NIF)]; if (idxNif==null){ for
(const k of aliasNif){ if (m[k]!=null){ idxNif=m[k]; break; } } }
    const aliasDir =
['direccionfacturacion','direcfacturacion','direccionfiscal','direcfiscal',
'direccion'];
```

```
    let idxDir = m[nn(FACT_MASTER.COL_BD_DIR)]; if (idxDir==null){ for
(const k of aliasDir){ if (m[k]!=null){ idxDir=m[k]; break; } } }
    const clean = s => String(s||'').replace(/\u00A0/g,' ').trim();
    const row = bd.find(r => nn(r[idxClienteBD]) === nn(cliente));
    if (row){ if (idxNif!=null) nifVal = clean(row[idxNif]); if
(idxDir!=null) dirVal = clean(row[idxDir]); }
  }
  meta.push(['nif', nifVal]);
  meta.push(['direccion_facturacion', dirVal]);
  shMeta.getRange(1,1,meta.length,2).setValues(meta);
  shMeta.autoResizeColumns(1,2);

  // Reordenar pestañas
  ssNuevo.setActiveSheet(shMeta);    ssNuevo.moveActiveSheet(1);
  ssNuevo.setActiveSheet(shResumen); ssNuevo.moveActiveSheet(2);
  ssNuevo.setActiveSheet(shLineas);  ssNuevo.moveActiveSheet(3);

  SpreadsheetApp.getActive().toast(`Archivo creado para Make: ${newName}`,
'Facturación', 6);
  return nuevo.getId();
}
```