/\*\*

\* =========================================================================

\* ملف: tools\_accounting.gs – محرك المحاسبة المتقدم v16.3

\* الوظيفة: تسجيل الإيرادات/المصروفات، والتحليل المالي، وتوليد تقارير

\* ديناميكية بصيغة JSON لدعم اتخاذ القرار.

\* =========================================================================

\*/

// --- 1. دوال مساعدة داخلية ---

function \_getSheet(tplKey) {

const tpl = getTemplates()[tplKey];

if (!tpl) throw new Error(`القالب "${tplKey}" غير موجود في env.gs`);

const ss = SpreadsheetApp.getActiveSpreadsheet();

let sheet = ss.getSheetByName(tpl.SHEET\_NAME);

if (!sheet) sheet = ss.insertSheet(tpl.SHEET\_NAME);

if (sheet.getLastRow() === 0) {

sheet.appendRow(tpl.HEADERS);

formatAsTable(sheet.getDataRange().getA1Notation(), sheet);

}

return sheet;

}

// --- 2. دوال تسجيل العمليات ---

function logRevenueEntry(row) {

logOperation('logRevenueEntry', 'start', row);

try {

const ENV = getEnvironment();

if (ENV === 'development') Logger.log('logRevenueEntry input: ' + JSON.stringify(row));

const sheet = \_getSheet('SALES');

const qty = Number(row.qty) || 0;

const unitPrice = Number(row.unitPrice) || 0;

if (qty <= 0 || unitPrice < 0) throw new Error("الكمية والسعر يجب أن تكون أرقامًا موجبة.");

const total = qty \* unitPrice;

const values = [new Date(row.date || Date.now()), row.code || 'N/A', row.product || 'منتج غير محدد', row.channel || 'صالة', qty, unitPrice, total];

sheet.appendRow(values);

logOperation('logRevenueEntry', 'done', { product: values[2], total });

return showMessage('success', `✅ تم تسجيل إيراد: ${values[2]} بقيمة ${total.toFixed(2)} ${getCurrencyCode()}`);

} catch (e) {

return showError(`فشل تسجيل الإيراد: ${e.message}`);

}

}

function logExpenseEntry(row) {

logOperation('logExpenseEntry', 'start', row);

try {

const ENV = getEnvironment();

if (ENV === 'development') Logger.log('logExpenseEntry input: ' + JSON.stringify(row));

const amount = Number(row.amount);

if (isNaN(amount) || amount <= 0) throw new Error("مبلغ المصروف يجب أن يكون رقمًا موجبًا.");

const sheet = \_getSheet('EXPENSES');

let category = row.category || 'غير مصنف';

const desc = (row.description || '').toLowerCase();

if (category === 'غير مصنف') {

const map = { 'خدمات ومرافق': /كهرباء|ماء|انترنت/, 'مصاريف إدارية': /إيجار|مكتب/, 'رواتب وأجور': /راتب|رواتب|مكافأة/ };

for (const [cat, rx] of Object.entries(map)) {

if (rx.test(desc)) { category = cat; break; }

}

}

const values = [new Date(row.date || Date.now()), row.description || '', amount, category];

sheet.appendRow(values);

logOperation('logExpenseEntry', 'done', { description: values[1], category, amount });

return showMessage('success', `✅ تم تسجيل مصروف: ${values[1]} بقيمة ${amount.toFixed(2)} ${getCurrencyCode()} (${category})`);

} catch (e) {

return showError(`فشل تسجيل المصروف: ${e.message}`);

}

}

// --- 3. دوال التحليل وتجميع البيانات ---

function calculateCategoryTotals(type, startDate = null, endDate = null) {

logOperation('calculateCategoryTotals', 'start', { type, startDate, endDate });

const key = type === 'Revenue' ? 'SALES' : 'EXPENSES';

const tpl = getTemplates()[key];

if (!tpl) throw new Error(`❌ القالب ${key} غير معرف في env.gs`);

const sheet = SpreadsheetApp.getActive().getSheetByName(tpl.sheetName);

if (!sheet || sheet.getLastRow() < 2) return { totalsByCategory: {}, totalAmount: 0 };

const data = sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn()).getValues();

const startMs = startDate ? new Date(startDate).setHours(0, 0, 0, 0) : null;

const endMs = endDate ? new Date(endDate).setHours(23, 59, 59, 999) : null;

const dateIdx = 0;

const amtIdx = key === 'SALES' ? 6 : 2;

const catIdx = key === 'SALES' ? 2 : 3;

const totals = {};

const total = data.reduce((acc, row) => {

const rowDate = new Date(row[dateIdx]);

if (isNaN(rowDate.getTime())) return acc;

if ((startMs && rowDate.getTime() < startMs) || (endMs && rowDate.getTime() > endMs)) return acc;

const amt = Number(row[amtIdx]) || 0;

const cat = row[catIdx] || 'غير مصنف';

totals[cat] = (totals[cat] || 0) + amt;

return acc + amt;

}, 0);

logOperation('calculateCategoryTotals', 'done', { total });

return { totalsByCategory: totals, totalAmount: total };

}

function calculateCategoryTotalsCached(type, startDate, endDate) {

const ENV = getEnvironment();

const USE\_CACHE = ENV === 'production';

if (!USE\_CACHE) return calculateCategoryTotals(type, startDate, endDate);

const key = `catTotals:${type}:${startDate || 'all'}:${endDate || 'all'}`;

const cache = CacheService.getScriptCache();

const c = cache.get(key);

if (c) return JSON.parse(c);

const res = calculateCategoryTotals(type, startDate, endDate);

cache.put(key, JSON.stringify(res), getCacheTTL());

return res;

}

// --- 4. دوال التقارير المالية ---

function calculateGrossProfit(startDate = null, endDate = null) {

logOperation('calculateGrossProfit', 'start', { startDate, endDate });

const rev = calculateCategoryTotalsCached('Revenue', startDate, endDate);

const exp = calculateCategoryTotalsCached('Expense', startDate, endDate);

const gp = rev.totalAmount - exp.totalAmount;

const out = { totalRevenue: rev.totalAmount, totalExpenses: exp.totalAmount, grossProfit: gp, currency: getCurrencyCode() };

logOperation('calculateGrossProfit', 'done', out);

return out;

}

function generateIncomeStatement(startDate = null, endDate = null) {

const totalsRev = calculateCategoryTotalsCached('Revenue', startDate, endDate);

const totalsExp = calculateCategoryTotalsCached('Expense', startDate, endDate);

return {

period: { startDate, endDate },

revenues: totalsRev.totalsByCategory,

expenses: totalsExp.totalsByCategory,

totalRevenue: totalsRev.totalAmount,

totalExpenses: totalsExp.totalAmount,

netIncome: totalsRev.totalAmount - totalsExp.totalAmount,

currency: getCurrencyCode()

};

}

function generateProfitabilityAnalysis(startDate = null, endDate = null) {

const totalsRev = calculateCategoryTotalsCached('Revenue', startDate, endDate);

const totalsExp = calculateCategoryTotalsCached('Expense', startDate, endDate);

const totalRevenue = totalsRev.totalAmount;

const analysis = {};

for (const [cat, amt] of Object.entries(totalsRev.totalsByCategory)) {

analysis[cat] = {

revenue: amt,

share: totalRevenue ? +(amt / totalRevenue \* 100).toFixed(2) : 0

};

}

return {

period: { startDate, endDate },

profitabilityByCategory: analysis,

totalRevenue,

currency: getCurrencyCode()

};

}

function generateDailySummary() {

logOperation('generateDailySummary', 'start');

const today = Utilities.formatDate(new Date(), Session.getScriptTimeZone(), "yyyy-MM-dd");

const data = calculateGrossProfit(today, today);

const summary = {

date: today,

revenue: data.totalRevenue,

expenses: data.totalExpenses,

net: data.grossProfit,

currency: data.currency

};

logOperation('generateDailySummary', 'done', summary);

return summary;

}

function detectAnomalies(type, factor = 2) {

logOperation('detectAnomalies', 'start', { type, factor });

const key = type === 'Revenue' ? 'SALES' : 'EXPENSES';

const tpl = getTemplates()[key];

const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(tpl.SHEET\_NAME);

if (!sheet || sheet.getLastRow() < 2) return [];

const rows = sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn()).getValues();

const amtIdx = key === 'SALES' ? 6 : 2;

const values = rows.map(r => Number(r[amtIdx]) || 0);

const mean = values.reduce((a, b) => a + b, 0) / values.length;

const variance = values.reduce((sum, v) => sum + Math.pow(v - mean, 2), 0) / values.length;

const stdDev = Math.sqrt(variance);

const thresholdLow = mean - factor \* stdDev;

const thresholdHigh = mean + factor \* stdDev;

const anomalies = rows

.map((row, i) => ({ row: i + 2, date: row[0], description: row[1], amount: row[amtIdx] }))

.filter(r => r.amount < thresholdLow || r.amount > thresholdHigh);

logOperation('detectAnomalies', 'done', { count: anomalies.length });

return anomalies;

}