/\*\*

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

\* @file tools\_accounting.gs – v16.3 (محرك المحاسبة المتقدم)

\* @module TOOLS\_ACCOUNTING

\* @description

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

\* ديناميكية بصيغة JSON لدعم اتخاذ القرار.

\* تعتمد على: HELPERS, AI\_TOOLS\_DIALOGUE, ENV, TOOLS\_SHEETS

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

\*/

var TOOLS\_ACCOUNTING = (function(HELPERS, AI\_TOOLS\_DIALOGUE, ENV, TOOLS\_SHEETS) { // ✅ حقن التبعيات

'use strict';

// --- 1. دوال مساعدة داخلية ---

/\*\*

\* @function \_getSheet

\* @description دالة مساعدة داخلية لجلب أو إنشاء ورقة عمل بناءً على قالب محدد.

\* تقوم بتهيئة الورقة بالرؤوس إذا كانت جديدة.

\* @param {string} tplKey - مفتاح القالب (مثلاً 'SALES', 'EXPENSES') من ENV.SETTINGS.TEMPLATES.

\* @returns {GoogleAppsScript.Spreadsheet.Sheet} - كائن الورقة.

\* @throws {Error} إذا كان القالب غير موجود.

\*/

function \_getSheet(tplKey) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', '\_getSheet:start', { tplKey });

const tpl = ENV.SETTINGS.TEMPLATES[tplKey]; // ✅ استخدام ENV.SETTINGS

if (!tpl) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', '\_getSheet:error', { tplKey, error: `القالب "${tplKey}" غير موجود في ENV.SETTINGS.TEMPLATES` });

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);

HELPERS.logOperation('TOOLS\_ACCOUNTING', '\_getSheet:created\_new\_sheet', { sheetName: tpl.SHEET\_NAME });

}

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

sheet.appendRow(tpl.HEADERS);

// ❌ تم إزالة استدعاء TOOLS\_SHEETS.formatAsTable

// هذه الدالة غير موجودة في TOOLS\_SHEETS.gs التي تم توفيرها.

// إذا كانت مطلوبة، يجب إضافتها صراحةً إلى TOOLS\_SHEETS.

HELPERS.logOperation('TOOLS\_ACCOUNTING', '\_getSheet:headers\_added', { sheetName: tpl.SHEET\_NAME });

}

HELPERS.logOperation('TOOLS\_ACCOUNTING', '\_getSheet:end', { sheetName: sheet.getName() });

return sheet;

}

// --- 2. دوال تسجيل العمليات ---

/\*\*

\* @function logRevenueEntry

\* @description تسجل إدخال إيراد جديد في ورقة المبيعات.

\* @param {{date?: string, code?: string, product?: string, channel?: string, qty: number, unitPrice: number}} row - كائن يحتوي على بيانات الإيراد.

\* @returns {object} - كائن رسالة نجاح أو خطأ من AI\_TOOLS\_DIALOGUE.

\*/

function logRevenueEntry(row) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:start', row);

try {

const ENV\_MODE = ENV.getEnvironment();

if (ENV\_MODE === 'development') HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:input', { row });

const sheet = \_getSheet('SALES');

const qty = Number(row.qty);

const unitPrice = Number(row.unitPrice);

if (isNaN(qty) || qty <= 0 || isNaN(unitPrice) || 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);

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:done', { product: values[2], total });

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

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:error', { fn: 'logRevenueEntry', error: e.message, input: row }); // ✅ تسجيل الخطأ

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

}

}

/\*\*

\* @function logExpenseEntry

\* @description تسجل إدخال مصروف جديد في ورقة المصروفات.

\* @param {{date?: string, description?: string, amount: number, category?: string}} row - كائن يحتوي على بيانات المصروف.

\* @returns {object} - كائن رسالة نجاح أو خطأ من AI\_TOOLS\_DIALOGUE.

\*/

function logExpenseEntry(row) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:start', row);

try {

const ENV\_MODE = ENV.getEnvironment();

if (ENV\_MODE === 'development') HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:input', { 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();

// منطق تحديد الفئة بناءً على الوصف

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);

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:done', { description: values[1], category, amount });

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

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:error', { fn: 'logExpenseEntry', error: e.message, input: row }); // ✅ تسجيل الخطأ

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

}

}

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

/\*\*

\* @function calculateCategoryTotals

\* @description تحسب إجمالي المبالغ حسب الفئة لنوع معين (إيرادات/مصروفات) ولفترة زمنية محددة.

\* @param {'Revenue'|'Expense'} type - نوع البيانات المراد حسابها ('Revenue' أو 'Expense').

\* @param {string|null} [startDate=null] - تاريخ البدء بصيغة "YYYY-MM-DD" (اختياري).

\* @param {string|null} [endDate=null] - تاريخ الانتهاء بصيغة "YYYY-MM-DD" (اختياري).

\* @returns {object} - كائن يحتوي على إجماليات حسب الفئة وإجمالي المبلغ الكلي.

\*/

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

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:start', { type, startDate, endDate });

try {

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

const tpl = ENV.SETTINGS.TEMPLATES[key];

if (!tpl) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:error', { fn: 'calculateCategoryTotals', error: `القالب "${key}" غير معرف في env.gs` });

return AI\_TOOLS\_DIALOGUE.showError(`❌ القالب ${key} غير معرف في env.gs`);

}

const sheet = \_getSheet(key);

if (!sheet || sheet.getLastRow() < 2) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:empty\_sheet', { sheetName: sheet ? sheet.getName() : 'N/A' });

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())) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:invalid\_date', { row: row[dateIdx] });

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);

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:done', { total });

return { totalsByCategory: totals, totalAmount: total };

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:error', { fn: 'calculateCategoryTotals', error: e.message, type, startDate, endDate }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل حساب إجماليات الفئات: ${e.message}`);

}

}

/\*\*

\* @function calculateCategoryTotalsCached

\* @description تحسب إجمالي المبالغ حسب الفئة وتخزن النتيجة مؤقتًا لزيادة الأداء.

\* @param {'Revenue'|'Expense'} type - نوع البيانات المراد حسابها.

\* @param {string|null} startDate - تاريخ البدء.

\* @param {string|null} endDate - تاريخ الانتهاء.

\* @returns {object} - كائن يحتوي على إجماليات حسب الفئة وإجمالي المبلغ الكلي.

\*/

function calculateCategoryTotalsCached(type, startDate, endDate) {

const ENV\_MODE = ENV.getEnvironment();

const USE\_CACHE = ENV\_MODE === '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) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotalsCached:from\_cache', { key });

return JSON.parse(c);

}

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

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

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotalsCached:to\_cache', { key });

return res;

}

// --- 4. دوال التقارير المالية ---

/\*\*

\* @function calculateGrossProfit

\* @description تحسب الربح الإجمالي لفترة زمنية محددة.

\* @param {string|null} [startDate=null] - تاريخ البدء بصيغة "YYYY-MM-DD" (اختياري).

\* @param {string|null} [endDate=null] - تاريخ الانتهاء بصيغة "YYYY-MM-DD" (اختياري).

\* @returns {object} - كائن يحتوي على إجمالي الإيرادات، إجمالي المصروفات، الربح الإجمالي، والعملة.

\*/

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

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateGrossProfit:start', { startDate, endDate });

try {

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

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

const gp = rev.totalAmount - exp.totalAmount;

const out = {

type: 'structured\_data', // ✅ إضافة نوع الإخراج المهيكل

title: 'تقرير الربح الإجمالي', // ✅ إضافة عنوان

data: {

totalRevenue: rev.totalAmount,

totalExpenses: exp.totalAmount,

grossProfit: gp,

currency: ENV.getCurrencyCode()

}

};

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateGrossProfit:done', out);

return out;

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateGrossProfit:error', { fn: 'calculateGrossProfit', error: e.message, startDate, endDate }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل حساب الربح الإجمالي: ${e.message}`);

}

}

/\*\*

\* @function generateIncomeStatement

\* @description تُولد بيان دخل (قائمة الأرباح والخسائر) مفصلًا لفترة زمنية محددة.

\* @param {string|null} [startDate=null] - تاريخ البدء بصيغة "YYYY-MM-DD" (اختياري).

\* @param {string|null} [endDate=null] - تاريخ الانتهاء بصيغة "YYYY-MM-DD" (اختياري).

\* @returns {object} - كائن يحتوي على تفاصيل بيان الدخل، بنوع إخراج مهيكل.

\*/

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

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateIncomeStatement:start', { startDate, endDate });

try {

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

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

const result = {

type: 'structured\_data', // ✅ إضافة نوع الإخراج المهيكل

title: 'بيان الدخل', // ✅ إضافة عنوان

data: {

period: { startDate, endDate },

revenues: totalsRev.totalsByCategory,

expenses: totalsExp.totalsByCategory,

totalRevenue: totalsRev.totalAmount,

totalExpenses: totalsExp.totalAmount,

netIncome: totalsRev.totalAmount - totalsExp.totalAmount,

currency: ENV.getCurrencyCode()

}

};

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateIncomeStatement:done', result);

return result;

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateIncomeStatement:error', { fn: 'generateIncomeStatement', error: e.message, startDate, endDate }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل توليد بيان الدخل: ${e.message}`);

}

}

/\*\*

\* @function generateProfitabilityAnalysis

\* @description تُولد تحليل ربحية حسب الفئة لفترة زمنية محددة.

\* @param {string|null} [startDate=null] - تاريخ البدء بصيغة "YYYY-MM-DD" (اختياري).

\* @param {string|null} [endDate=null] - تاريخ الانتهاء بصيغة "YYYY-MM-DD" (اختياري).

\* @returns {object} - كائن يحتوي على تحليل الربحية حسب الفئة، بنوع إخراج مهيكل.

\*/

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

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateProfitabilityAnalysis:start', { startDate, endDate });

try {

const totalsRev = calculateCategoryTotalsCached('Revenue', 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

};

}

const result = {

type: 'structured\_data', // ✅ إضافة نوع الإخراج المهيكل

title: 'تحليل الربحية حسب الفئة', // ✅ إضافة عنوان

data: {

period: { startDate, endDate },

profitabilityByCategory: analysis,

totalRevenue,

currency: ENV.getCurrencyCode()

}

};

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateProfitabilityAnalysis:done', result);

return result;

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateProfitabilityAnalysis:error', { fn: 'generateProfitabilityAnalysis', error: e.message, startDate, endDate }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل توليد تحليل الربحية: ${e.message}`);

}

}

/\*\*

\* @function generateDailySummary

\* @description تُولد ملخصًا يوميًا للإيرادات والمصروفات وصافي الربح لليوم الحالي.

\* @returns {object} - كائن يحتوي على الملخص اليومي، بنوع إخراج مهيكل.

\*/

function generateDailySummary() {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateDailySummary:start');

try {

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

const data = calculateGrossProfit(today, today).data; // ✅ جلب البيانات من الربح الإجمالي المهيكل

const summary = {

type: 'structured\_data', // ✅ إضافة نوع الإخراج المهيكل

title: 'الملخص اليومي', // ✅ إضافة عنوان

data: {

date: today,

revenue: data.totalRevenue,

expenses: data.totalExpenses,

net: data.grossProfit,

currency: data.currency

}

};

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateDailySummary:done', summary);

return summary;

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateDailySummary:error', { fn: 'generateDailySummary', error: e.message }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل توليد الملخص اليومي: ${e.message}`);

}

}

/\*\*

\* @function detectAnomalies

\* @description تكتشف القيم الشاذة (Anomalies) في بيانات الإيرادات أو المصروفات باستخدام الانحراف المعياري.

\* @param {'Revenue'|'Expense'} type - نوع البيانات المراد تحليلها ('Revenue' أو 'Expense').

\* @param {number} [factor=2] - عامل الانحراف المعياري لتحديد الشذوذ.

\* @returns {Array<object>|object} - مصفوفة من الكائنات التي تمثل الصفوف الشاذة، أو كائن رسالة خطأ.

\*/

function detectAnomalies(type, factor = 2) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:start', { type, factor });

try {

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

const tpl = ENV.SETTINGS.TEMPLATES[key];

if (!tpl) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:error', { fn: 'detectAnomalies', error: `القالب "${key}" غير معرف في env.gs` });

return AI\_TOOLS\_DIALOGUE.showError(`❌ القالب ${key} غير معرف في env.gs`);

}

const sheet = \_getSheet(key);

if (!sheet || sheet.getLastRow() < 2) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:empty\_sheet', { sheetName: sheet ? sheet.getName() : 'N/A' });

return []; // إرجاع مصفوفة فارغة إذا كانت الورقة فارغة

}

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);

if (values.length === 0) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:no\_values', { type });

return [];

}

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);

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:done', { count: anomalies.length });

return AI\_TOOLS\_DIALOGUE.showSummaryTable(anomalies, `القيم الشاذة في ${type} (عامل ${factor})`); // ✅ عرض النتائج في جدول ملخص

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'detectAnomalies:error', { fn: 'detectAnomalies', error: e.message, type, factor }); // ✅ تسجيل الخطأ

return AI\_TOOLS\_DIALOGUE.showError(`فشل اكتشاف القيم الشاذة: ${e.message}`);

}

}

// --- تصدير الواجهة العامة للوحدة ---

return {

logRevenueEntry,

logExpenseEntry,

calculateCategoryTotals,

calculateGrossProfit,

generateIncomeStatement,

generateProfitabilityAnalysis,

generateDailySummary,

detectAnomalies

};

})(HELPERS, AI\_TOOLS\_DIALOGUE, ENV, TOOLS\_SHEETS);