/\*\*

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

\* @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.getTemplates()[tplKey ]; // ✅ استخدام ENV.getTemplates()

if (!tpl) {

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

throw new Error(`القالب "${tplKey}" غير موجود في إعدادات البيئة.`);

}

const ss = SpreadsheetApp.getActiveSpreadsheet();

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

if (!sheet) {

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

if (tpl.HEADERS && tpl.HEADERS.length > 0) {

sheet.appendRow(tpl.HEADERS);

HELPERS.formatAsTable(sheet.getDataRange().getA1Notation(), sheet); // تنسيق كجدول

}

}

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

return sheet;

}

// --- 2. دوال تسجيل العمليات ---

/\*\*

\* @function logRevenueEntry

\* @description

\* تسجل إدخال إيراد جديد في ورقة المبيعات.

\* @param {object} rowData - كائن يحتوي على بيانات الإيراد (product, qty, price, discount, notes).

\* @returns {object} - رسالة نجاح أو خطأ.

\*/

function logRevenueEntry(rowData) {

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

try {

const sheet = \_getSheet('SALES');

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

const price = Number(rowData.price) || 0;

const discount = Number(rowData.discount) || 0;

const total = (qty \* price) \* (1 - discount);

const currency = ENV.getCurrencyCode(); // ✅ استخدام ENV.getCurrencyCode()

sheet.appendRow([

Utilities.formatDate(new Date(), Session.getScriptTimeZone(), ENV.getDateFormat()), // ✅ استخدام ENV.getDateFormat()

rowData.product,

qty,

price,

discount,

total,

currency,

rowData.notes || ''

]);

HELPERS.formatAsTable(sheet.getDataRange().getA1Notation(), sheet); // إعادة تنسيق الجدول

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:success', { product: rowData.product, total });

return AI\_TOOLS\_DIALOGUE.showMessage('success',

`✅ تم تسجيل إيراد: ${rowData.product} ×${qty} بقيمة ${total.toFixed(2)} ${currency}`);

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logRevenueEntry:error', { error: e.message, rowData });

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

}

}

/\*\*

\* @function logExpenseEntry

\* @description

\* تسجل إدخال مصروف جديد في ورقة المصروفات.

\* @param {object} rowData - كائن يحتوي على بيانات المصروف (description, amount, category, notes).

\* @returns {object} - رسالة نجاح أو خطأ.

\*/

function logExpenseEntry(rowData) {

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

try {

const sheet = \_getSheet('EXPENSES');

const amount = Number(rowData.amount) || 0;

const currency = ENV.getCurrencyCode(); // ✅ استخدام ENV.getCurrencyCode()

sheet.appendRow([

Utilities.formatDate(new Date(), Session.getScriptTimeZone(), ENV.getDateFormat()), // ✅ استخدام ENV.getDateFormat()

rowData.description,

amount,

rowData.category || 'عام',

currency,

rowData.notes || ''

]);

HELPERS.formatAsTable(sheet.getDataRange().getA1Notation(), sheet); // إعادة تنسيق الجدول

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:success', { description: rowData.description, amount });

return AI\_TOOLS\_DIALOGUE.showMessage('success',

`✅ تم تسجيل مصروف: ${rowData.description} بقيمة ${amount.toFixed(2)} ${currency}`);

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'logExpenseEntry:error', { error: e.message, rowData });

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

}

}

// --- 3. دوال التحليل والتقارير ---

/\*\*

\* @function calculateGrossProfit

\* @description

\* يحسب إجمالي الربح لفترة زمنية محددة.

\* @param {string} startDate - تاريخ البدء (بصيغة YYYY-MM-DD).

\* @param {string} endDate - تاريخ الانتهاء (بصيغة YYYY-MM-DD).

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

\*/

function calculateGrossProfit(startDate, endDate) {

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

try {

const salesSheet = \_getSheet('SALES');

const expensesSheet = \_getSheet('EXPENSES');

const dateFormat = ENV.getDateFormat();

const currency = ENV.getCurrencyCode();

let totalRevenue = 0;

if (salesSheet.getLastRow() > 1) {

const salesData = salesSheet.getRange(2, 1, salesSheet.getLastRow() - 1, salesSheet.getLastColumn()).getValues();

salesData.forEach(row => {

const date = Utilities.formatDate(row[0], Session.getScriptTimeZone(), dateFormat);

if (date >= startDate && date <= endDate) {

totalRevenue += Number(row[5]) || 0; // عمود الإجمالي

}

});

}

let totalExpenses = 0;

if (expensesSheet.getLastRow() > 1) {

const expensesData = expensesSheet.getRange(2, 1, expensesSheet.getLastRow() - 1, expensesSheet.getLastColumn()).getValues();

expensesData.forEach(row => {

const date = Utilities.formatDate(row[0], Session.getScriptTimeZone(), dateFormat);

if (date >= startDate && date <= endDate) {

totalExpenses += Number(row[2]) || 0; // عمود المبلغ

}

});

}

const grossProfit = totalRevenue - totalExpenses;

const result = {

totalRevenue: totalRevenue.toFixed(2),

totalExpenses: totalExpenses.toFixed(2),

grossProfit: grossProfit.toFixed(2),

currency: currency

};

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateGrossProfit:success', result);

return AI\_TOOLS\_DIALOGUE.showSummaryTable([

{ Metric: "إجمالي الإيرادات", Value: `${result.totalRevenue} ${currency}` },

{ Metric: "إجمالي المصروفات", Value: `${result.totalExpenses} ${currency}` },

{ Metric: "صافي الربح", Value: `${result.grossProfit} ${currency}` }

], `صافي الربح للفترة من ${startDate} إلى ${endDate}`);

} catch (e) {

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

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

}

}

/\*\*

\* @function generateDailySummary

\* @description

\* يولد ملخصًا يوميًا للإيرادات والمصروفات وصافي الربح ويحفظه في ورقة الملخص اليومي.

\* @param {string} [date] - التاريخ المحدد (بصيغة YYYY-MM-DD)، الافتراضي هو اليوم.

\* @returns {object} - كائن الملخص اليومي أو رسالة خطأ.

\*/

function generateDailySummary(date) {

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

try {

const today = date || Utilities.formatDate(new Date(), Session.getScriptTimeZone(), ENV.getDateFormat());

const data = calculateGrossProfit(today, today); // تستدعي الدالة السابقة

const summarySheet = \_getSheet('DAILY\_SUMMARY');

const currency = ENV.getCurrencyCode();

const newRow = [

today,

Number(data.totalRevenue),

Number(data.totalExpenses),

Number(data.grossProfit),

currency

];

// التحقق مما إذا كان هناك صف لهذا التاريخ بالفعل وتحديثه

const existingData = summarySheet.getDataRange().getValues();

let found = false;

for (let i = 1; i < existingData.length; i++) {

if (Utilities.formatDate(existingData[i][0], Session.getScriptTimeZone(), ENV.getDateFormat()) === today) {

summarySheet.getRange(i + 1, 1, 1, newRow.length).setValues([newRow]);

found = true;

break;

}

}

if (!found) {

summarySheet.appendRow(newRow);

}

HELPERS.formatAsTable(summarySheet.getDataRange().getA1Notation(), summarySheet); // إعادة تنسيق الجدول

const result = {

date: today,

revenue: data.totalRevenue,

expenses: data.totalExpenses,

net: data.grossProfit,

currency: currency

};

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

return AI\_TOOLS\_DIALOGUE.showSummaryTable([

{ Metric: "التاريخ", Value: result.date },

{ Metric: "إجمالي الإيرادات", Value: `${result.revenue} ${currency}` },

{ Metric: "إجمالي المصروفات", Value: `${result.expenses} ${currency}` },

{ Metric: "صافي الربح", Value: `${result.net} ${currency}` }

], `ملخص يومي لتاريخ ${result.date}`);

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'generateDailySummary:error', { error: e.message, date });

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

}

}

/\*\*

\* @function detectAnomalies

\* @description

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

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

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

\* @returns {object} - جدول ملخص بالقيم الشاذة أو رسالة معلومات.

\*/

function detectAnomalies(type, factor = 2) {

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

try {

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

const sheet = \_getSheet(key);

if (sheet.getLastRow() < 2) {

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

return AI\_TOOLS\_DIALOGUE.showMessage("info", `ℹ️ ورقة "${sheet.getName()}" فارغة، لا توجد بيانات لاكتشاف الشذوذ.`);

}

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

const amtIdx = key === 'SALES' ? 5 : 2; // عمود الإجمالي للمبيعات، وعمود المبلغ للمصروفات

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: Utilities.formatDate(row[0], Session.getScriptTimeZone(), ENV.getDateFormat()), description: row[1], amount: row[amtIdx] }))

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

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

if (anomalies.length > 0) {

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

} else {

return AI\_TOOLS\_DIALOGUE.showMessage("success", `✅ لم يتم العثور على قيم شاذة في ${type} باستخدام عامل ${factor}.`);

}

} 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}`);

}

}

/\*\*

\* @function calculateCategoryTotals

\* @description

\* يحسب إجمالي المبالغ لكل فئة من المصروفات.

\* @returns {object} - جدول ملخص بإجمالي المصروفات حسب الفئة أو رسالة خطأ.

\*/

function calculateCategoryTotals() {

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

try {

const expensesSheet = \_getSheet('EXPENSES');

if (expensesSheet.getLastRow() < 2) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:empty\_sheet', {});

return AI\_TOOLS\_DIALOGUE.showMessage("info", "ℹ️ ورقة المصروفات فارغة، لا توجد بيانات لحساب الإجماليات.");

}

const expensesData = expensesSheet.getRange(2, 1, expensesSheet.getLastRow() - 1, expensesSheet.getLastColumn()).getValues();

const categoryTotals = {};

const currency = ENV.getCurrencyCode();

expensesData.forEach(row => {

const category = row[3] || 'غير مصنف'; // عمود الفئة

const amount = Number(row[2]) || 0; // عمود المبلغ

categoryTotals[category] = (categoryTotals[category] || 0) + amount;

});

const result = Object.keys(categoryTotals).map(category => ({

Category: category,

Total: `${categoryTotals[category].toFixed(2)} ${currency}`

}));

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:success', { count: result.length });

return AI\_TOOLS\_DIALOGUE.showSummaryTable(result, "إجمالي المصروفات حسب الفئة");

} catch (e) {

HELPERS.logOperation('TOOLS\_ACCOUNTING', 'calculateCategoryTotals:error', { error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`❌ فشل حساب إجمالي المصروفات حسب الفئة: ${e.message}`);

}

}

/\*\*

\* @function generateIncomeStatement

\* @description

\* يولد بيان دخل مبسط (قائمة الدخل) لفترة زمنية محددة.

\* @param {string} startDate - تاريخ البدء (بصيغة YYYY-MM-DD).

\* @param {string} endDate - تاريخ الانتهاء (بصيغة YYYY-MM-DD).

\* @returns {object} - جدول ملخص ببيان الدخل أو رسالة خطأ.

\*/

function generateIncomeStatement(startDate, endDate) {

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

try {

const salesSheet = \_getSheet('SALES');

const expensesSheet = \_getSheet('EXPENSES');

const dateFormat = ENV.getDateFormat();

const currency = ENV.getCurrencyCode();

let totalRevenue = 0;

if (salesSheet.getLastRow() > 1) {

const salesData = salesSheet.getRange(2, 1, salesSheet.getLastRow() - 1, salesSheet.getLastColumn()).getValues();

salesData.forEach(row => {

const date = Utilities.formatDate(row[0], Session.getScriptTimeZone(), dateFormat);

if (date >= startDate && date <= endDate) {

totalRevenue += Number(row[5]) || 0;

}

});

}

let totalExpenses = 0;

const expensesByCategory = {};

if (expensesSheet.getLastRow() > 1) {

const expensesData = expensesSheet.getRange(2, 1, expensesSheet.getLastRow() - 1, expensesSheet.getLastColumn()).getValues();

expensesData.forEach(row => {

const date = Utilities.formatDate(row[0], Session.getScriptTimeZone(), dateFormat);

if (date >= startDate && date <= endDate) {

const category = row[3] || 'غير مصنف';

const amount = Number(row[2]) || 0;

totalExpenses += amount;

expensesByCategory[category] = (expensesByCategory[category] || 0) + amount;

}

});

}

const netIncome = totalRevenue - totalExpenses;

const incomeStatement = [];

incomeStatement.push({ Item: "إجمالي الإيرادات", Amount: `${totalRevenue.toFixed(2)} ${currency}` });

incomeStatement.push({ Item: "إجمالي المصروفات", Amount: `${totalExpenses.toFixed(2)} ${currency}` });

Object.keys(expensesByCategory).forEach(category => {

incomeStatement.push({ Item: ` - ${category}`, Amount: `${expensesByCategory[category].toFixed(2)} ${currency}` });

});

incomeStatement.push({ Item: "صافي الدخل", Amount: `${netIncome.toFixed(2)} ${currency}` });

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

return AI\_TOOLS\_DIALOGUE.showSummaryTable(incomeStatement, `بيان الدخل للفترة من ${startDate} إلى ${endDate}`);

} catch (e) {

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

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

}

}

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

return {

logRevenueEntry,

logExpenseEntry,

calculateGrossProfit,

generateDailySummary,

detectAnomalies,

calculateCategoryTotals,

generateIncomeStatement

};

})(

GAssistant.Helpers,

GAssistant.AI.Dialogue,

GAssistant.Config, // تم تغيير ENV إلى GAssistant.Config

GAssistant.Tools.Sheets // افتراض وجود هذه الوحدة

);