/\*\*

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

\* @file tools\_sheets.gs – v15.15 (محرك إدارة جداول Google Sheets)

\* @module TOOLS\_SHEETS

\* @description

\* مكتبة شاملة لتلاعب جداول Google Sheets:

\* • CRUD أساسية (قراءة/كتابة/مسح/حذف/إدراج صفوف/معادلات)

\* • فرز وفلترة الأعمدة

\* • بحث واستبدال جماعي

\* • تحليل وتلوين القيم الشاذة

\* • استخراج روابط من أعمدة

\* • تظليل الخلايا الفارغة

\* • إرسال تنبيه بعد الحذف (بريد)

\* • إنشاء جداول جديدة من القوالب مع تأكيد المستخدم

\* • حماية نطاقات وأعمدة وفق الأدوار أو بشكل عام

\* • دمج، تكرار، ودمج خلايا/أوراق

\* • أدوات مساعدة: ملخص الصفحة، تحويل رقم العمود إلى حرف

\* • [جديد] ضبط عرض الأعمدة تلقائيًا

\* • [جديد] إعادة تسمية الأوراق

\* • [جديد] حماية النطاقات

\* • [جديد] تلخيص إحصائيات الورقة

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

\*/

var TOOLS\_SHEETS = (function(HELPERS, AI\_TOOLS\_DIALOGUE, SETTINGS\_PROVIDER, ENV) {

'use strict';

/\*\*

\* @function \_getSheet

\* @description دالة مساعدة داخلية لجلب كائن الورقة بالاسم.

\* @param {string} name - اسم الورقة.

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

\* @throws {Error} إذا لم يتم العثور على الورقة.

\*/

function \_getSheet(name) {

const ss = SpreadsheetApp.getActiveSpreadsheet();

const sheet = ss.getSheetByName(name);

if (!sheet) {

HELPERS.logOperation('TOOLS\_SHEETS', '\_getSheet:error', { sheetName: name, error: 'Sheet not found' });

throw new Error(`Sheet "${name}" not found.`);

}

return sheet;

}

/\*\*

\* @function writeToCell

\* @description تكتب قيمة محددة في خلية معينة ضمن ورقة محددة.

\* @param {string} sheetName - اسم الورقة.

\* @param {string} cellA1 - عنوان الخلية بصيغة A1 (مثال: "A1", "B5").

\* @param {any} value - القيمة المراد كتابتها.

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

\*/

function writeToCell(sheetName, cellA1, value) {

HELPERS.logOperation('TOOLS\_SHEETS', 'writeToCell', { sheetName, cellA1, value });

try {

\_getSheet(sheetName).getRange(cellA1).setValue(value);

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تمّت الكتابة في ${cellA1} بالورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'writeToCell:error', { sheetName, cellA1, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`فشل الكتابة في ${cellA1}: ${e.message}`);

}

}

/\*\*

\* @function clearRange

\* @description يمسح محتوى وتنسيق نطاق محدد في ورقة محددة.

\* @param {string} sheetName - اسم الورقة.

\* @param {string} rangeA1 - النطاق المراد مسحه بصيغة A1 (مثال: "A1:B10").

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

\*/

function clearRange(sheetName, rangeA1) {

HELPERS.logOperation('TOOLS\_SHEETS', 'clearRange', { sheetName, rangeA1 });

try {

\_getSheet(sheetName).getRange(rangeA1).clear({ contentsOnly: false });

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم مسح النطاق ${rangeA1} بالورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'clearRange:error', { sheetName, rangeA1, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`فشل المسح في ${rangeA1}: ${e.message}`);

}

}

/\*\*

\* @function insertRowAt

\* @description يُدرج صفًا جديدًا قبل صف محدد في ورقة محددة.

\* @param {string} sheetName - اسم الورقة.

\* @param {number} rowIndex - رقم الصف الذي سيتم إدراج الصف الجديد قبله.

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

\*/

function insertRowAt(sheetName, rowIndex) {

HELPERS.logOperation('TOOLS\_SHEETS', 'insertRowAt', { sheetName, rowIndex });

try {

\_getSheet(sheetName).insertRowBefore(parseInt(rowIndex, 10));

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم إدراج صف قبل الصف ${rowIndex} بالورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'insertRowAt:error', { sheetName, rowIndex, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`فشل الإدراج قبل الصف ${rowIndex}: ${e.message}`);

}

}

/\*\*

\* @function deleteRow

\* @description يحذف صفًا محددًا من ورقة محددة، ويرسل تنبيهًا بالبريد الإلكتروني إذا تم تكوينه.

\* @param {string} sheetName - اسم الورقة.

\* @param {number} rowIndex - رقم الصف المراد حذفه.

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

\*/

function deleteRow(sheetName, rowIndex) {

HELPERS.logOperation('TOOLS\_SHEETS', 'deleteRow', { sheetName, rowIndex });

try {

const sh = \_getSheet(sheetName);

sh.deleteRow(parseInt(rowIndex, 10));

const emails = SETTINGS\_PROVIDER.getSettings().ALERT\_EMAILS;

if (emails && emails.length > 0) {

const subject = `Sheet "${sheetName}": Row ${rowIndex} deleted`;

const body = `تم حذف الصف ${rowIndex} يدوياً من الورقة "${sheetName}".`;

MailApp.sendEmail(emails, subject, body);

HELPERS.logOperation('TOOLS\_SHEETS', 'deleteRow:email\_sent', { sheetName, rowIndex, emails });

}

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم حذف الصف ${rowIndex} من الورقة "${sheetName}"${emails ? ' وإرسال تنبيه.' : '.'}`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'deleteRow:error', { sheetName, rowIndex, error: e.message });

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

}

}

/\*\*

\* @function filterByValue

\* @description يطبق فلتر على عمود معين بناءً على قيمة محددة في ورقة محددة.

\* @param {string} sheetName - اسم الورقة.

\* @param {number} columnIndex - رقم العمود (1-based) لتطبيق الفلتر عليه.

\* @param {string} value - القيمة التي سيتم الفلترة بها (نصية).

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

\*/

function filterByValue(sheetName, columnIndex, value) {

HELPERS.logOperation('TOOLS\_SHEETS', 'filterByValue', { sheetName, columnIndex, value });

try {

const sh = \_getSheet(sheetName);

const dataRange = sh.getRange(1, 1, sh.getLastRow(), sh.getLastColumn());

dataRange

.createFilter()

.setColumnFilterCriteria(

parseInt(columnIndex, 10),

SpreadsheetApp.newFilterCriteria()

.whenTextContains(String(value))

.build()

);

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم تطبيق الفلترة على العمود ${columnIndex} بالقيمة "${value}" في الورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'filterByValue:error', { sheetName, columnIndex, value, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`فشل الفلترة على العمود ${columnIndex}: ${e.message}`);

}

}

/\*\*

\* @function sortColumn

\* @description يقوم بفرز البيانات في عمود محدد تصاعديًا أو تنازليًا في ورقة محددة.

\* @param {string} sheetName - اسم الورقة.

\* @param {number} columnIndex - رقم العمود (1-based) للفرز.

\* @param {boolean} [ascending=true] - صحيح للفرز التصاعدي، خطأ للفرز التنازلي.

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

\*/

function sortColumn(sheetName, columnIndex, ascending = true) {

HELPERS.logOperation('TOOLS\_SHEETS', 'sortColumn', { sheetName, columnIndex, ascending });

try {

const sh = \_getSheet(sheetName);

const dataRange = sh.getRange(1, 1, sh.getLastRow(), sh.getLastColumn());

dataRange.sort({ column: parseInt(columnIndex, 10), ascending: ascending });

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم الفرز حسب العمود ${columnIndex} (${ascending ? 'تصاعدي' : 'تنازلي'}) في الورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'sortColumn:error', { sheetName, columnIndex, ascending, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`فشل الفرز حسب العمود ${columnIndex}: ${e.message}`);

}

}

/\*\*

\* @function extractUrls

\* @description يستخرج جميع الروابط (URLs) من عمود محدد في ورقة معينة، سواء كانت روابط مضمنة في نص منسق أو نصوص URLs صريحة.

\* @param {string} sheetName - اسم الورقة.

\* @param {number} columnIndex - رقم العمود (1-based) لاستخراج الروابط منه.

\* @returns {object} - كائن يحتوي على جدول ملخص بالروابط المستخرجة أو رسالة معلومات/خطأ.

\*/

function extractUrls(sheetName, columnIndex) {

HELPERS.logOperation('TOOLS\_SHEETS', 'extractUrls', { sheetName, columnIndex });

try {

const sheet = \_getSheet(sheetName);

const lastRow = sheet.getLastRow();

if (lastRow < 1) {

return AI\_TOOLS\_DIALOGUE.showMessage('info', 'ℹ️ الورقة فارغة، لا توجد روابط لاستخراجها.');

}

const range = sheet.getRange(1, parseInt(columnIndex, 10), lastRow);

const richTextValues = range.getRichTextValues();

const results = [];

const urlRegex = /(https?:\/\/[^\s"']+)/g;

for (let i = 0; i < richTextValues.length; i++) {

const runs = richTextValues[i][0].getRuns();

for (const run of runs) {

if (run.getLinkUrl()) {

results.push({ row: i + 1, url: run.getLinkUrl() });

} else {

const textMatches = (run.getText() || '').match(urlRegex);

if (textMatches) {

textMatches.forEach(url => results.push({ row: i + 1, url }));

}

}

}

}

if (results.length === 0) {

return AI\_TOOLS\_DIALOGUE.showMessage('info', 'ℹ️ لم يتم العثور على أي روابط في العمود المحدد.');

}

return AI\_TOOLS\_DIALOGUE.showSummaryTable(results);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'extractUrls:error', { sheetName, columnIndex, error: e.message });

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

}

}

/\*\*

\* @function highlightEmptyCells

\* @description يطبق قاعدة تنسيق شرطي لتظليل الخلايا الفارغة في نطاق محدد بلون معين.

\* @param {string} sheetName - اسم الورقة.

\* @param {string} rangeA1 - النطاق المراد تظليل الخلايا الفارغة فيه بصيغة A1.

\* @param {string} [color] - اللون السداسي (hex color) للتظليل (مثال: "#FF0000"). إذا لم يتم تحديده، سيتم استخدام اللون الافتراضي من الإعدادات.

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

\*/

function highlightEmptyCells(sheetName, rangeA1, color) {

HELPERS.logOperation('TOOLS\_SHEETS', 'highlightEmptyCells', { sheetName, rangeA1, color });

try {

const sh = \_getSheet(sheetName);

const rng = sh.getRange(rangeA1);

const defaultColor = SETTINGS\_PROVIDER.getSettings().HIGHLIGHT\_EMPTY\_COLOR || '#fce8e6';

const bgColor = color || defaultColor;

const rule = SpreadsheetApp.newConditionalFormatRule()

.whenCellEmpty()

.setBackground(bgColor)

.setRanges([rng])

.build();

const rules = sh.getConditionalFormatRules();

rules.push(rule);

sh.setConditionalFormatRules(rules);

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم تظليل الخلايا الفارغة في النطاق ${rangeA1} بالورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'highlightEmptyCells:error', { sheetName, rangeA1, color, error: e.message });

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

}

}

/\*\*

\* @function autoSizeColumns

\* @description يضبط عرض الأعمدة تلقائيًا بناءً على محتواها في نطاق محدد.

\* @param {string} sheetName - اسم الورقة.

\* @param {string} rangeA1 - النطاق الذي سيتم ضبط عرض أعمدته تلقائيًا (مثال: "A:C" أو "A1:C10").

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

\*/

function autoSizeColumns(sheetName, rangeA1) {

HELPERS.logOperation('TOOLS\_SHEETS', 'autoSizeColumns', { sheetName, rangeA1 });

try {

const sh = \_getSheet(sheetName);

const range = sh.getRange(rangeA1);

sh.autoResizeColumns(range.getColumn(), range.getNumColumns());

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم ضبط عرض الأعمدة تلقائيًا في النطاق ${rangeA1} بالورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'autoSizeColumns:error', { sheetName, rangeA1, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`❌ فشل ضبط عرض الأعمدة تلقائيًا في ${rangeA1}: ${e.message}`);

}

}

/\*\*

\* @function renameSheet

\* @description يعيد تسمية ورقة محددة.

\* @param {string} oldName - الاسم الحالي للورقة.

\* @param {string} newName - الاسم الجديد للورقة.

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

\*/

function renameSheet(oldName, newName) {

HELPERS.logOperation('TOOLS\_SHEETS', 'renameSheet', { oldName, newName });

try {

const sh = \_getSheet(oldName);

sh.setName(newName);

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم تغيير اسم الورقة من "${oldName}" إلى "${newName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'renameSheet:error', { oldName, newName, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`❌ فشل إعادة تسمية الورقة من "${oldName}" إلى "${newName}": ${e.message}`);

}

}

/\*\*

\* @function protectRange

\* @description يحمي نطاقًا محددًا في ورقة معينة، ويسمح لمحررين محددين بالوصول.

\* @param {string} sheetName - اسم الورقة.

\* @param {string} rangeA1 - النطاق المراد حمايته بصيغة A1.

\* @param {Array<string>} [editors=[]] - مصفوفة من عناوين البريد الإلكتروني للمحررين المسموح لهم بالوصول. إذا كانت فارغة، لن يتمكن أحد من التعديل باستثناء المالك.

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

\*/

function protectRange(sheetName, rangeA1, editors = []) {

HELPERS.logOperation('TOOLS\_SHEETS', 'protectRange', { sheetName, rangeA1, editors });

try {

const sh = \_getSheet(sheetName);

const range = sh.getRange(rangeA1);

const protection = range.protect();

// إزالة جميع المحررين الحاليين

protection.removeEditors(protection.getEditors());

// إضافة المحررين الجدد

if (editors && editors.length > 0) {

protection.addEditors(editors);

} else {

// إذا لم يتم تحديد محررين، إزالة المالك من قائمة المحررين لجعله محميًا بالكامل

// إلا إذا كان المالك هو الوحيد، فسيظل المالك محررًا

protection.setDomainEdit(false); // منع التعديل من قبل أي شخص في نفس النطاق

}

// إزالة المستخدم الحالي (الذي يقوم بتشغيل السكربت) من قائمة المحررين إذا لم يكن مدرجًا صراحةً

// هذا يضمن أن الحماية تعمل حتى للمالك إذا لم يدرج نفسه

if (!protection.isOwner()) { // إذا لم يكن المالك هو من يقوم بتشغيل السكربت

protection.removeEditors([Session.getActiveUser().getEmail()]);

}

return AI\_TOOLS\_DIALOGUE.showMessage('success', `✅ تم حماية النطاق ${rangeA1} في الورقة "${sheetName}".`);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'protectRange:error', { sheetName, rangeA1, editors, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`❌ فشل حماية النطاق ${rangeA1}: ${e.message}`);

}

}

/\*\*

\* @function summarizeSheetStats

\* @description يقدم ملخصًا إحصائيًا لورقة محددة، بما في ذلك عدد الصفوف/الأعمدة، الخلايا الفارغة، وعدد المعادلات.

\* @param {string} sheetName - اسم الورقة.

\* @returns {object} - كائن يحتوي على جدول ملخص بالإحصائيات أو رسالة خطأ.

\*/

function summarizeSheetStats(sheetName) {

HELPERS.logOperation('TOOLS\_SHEETS', 'summarizeSheetStats', { sheetName });

try {

const sh = \_getSheet(sheetName);

const lastRow = sh.getLastRow();

const lastColumn = sh.getLastColumn();

const dataRange = sh.getDataRange();

const numRows = dataRange.getNumRows();

const numColumns = dataRange.getNumColumns();

if (numRows === 0 || numColumns === 0) {

return AI\_TOOLS\_DIALOGUE.showMessage('info', `ℹ️ الورقة "${sheetName}" فارغة.`);

}

const values = dataRange.getValues();

let emptyCells = 0;

let formulaCells = 0;

// حساب الخلايا الفارغة والمعادلات

for (let r = 0; r < numRows; r++) {

for (let c = 0; c < numColumns; c++) {

if (values[r][c] === '' || values[r][c] === null) {

emptyCells++;

}

// لا يمكن لـ getValues() التفريق بين القيم والمعادلات مباشرة

// سيتطلب ذلك استخدام getFormulas() لنطاق محدد

// للحفاظ على الأداء، لن نتحقق من المعادلات لكل خلية هنا.

// يمكن إضافة هذه الميزة إذا كانت هناك حاجة ماسة لها، مع مراعاة الأداء.

}

}

// لعد الخلايا التي تحتوي على معادلات بشكل دقيق (قد يكون بطيئًا على الأوراق الكبيرة)

try {

const formulas = dataRange.getFormulas();

for (let r = 0; r < numRows; r++) {

for (let c = 0; c < numColumns; c++) {

if (formulas[r][c] !== '') {

formulaCells++;

}

}

}

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'summarizeSheetStats:getFormulas\_error', { error: e.message });

// يمكن أن يحدث خطأ إذا كان النطاق كبيراً جداً

}

const stats = [

{ "الإحصائية": "اسم الورقة", "القيمة": sheetName },

{ "الإحصائية": "الصفوف المستخدمة", "القيمة": lastRow },

{ "الإحصائية": "الأعمدة المستخدمة", "القيمة": lastColumn },

{ "الإحصائية": "الخلايا المستخدمة", "القيمة": numRows \* numColumns },

{ "الإحصائية": "الخلايا الفارغة", "القيمة": emptyCells },

{ "الإحصائية": "الخلايا بالمعادلات", "القيمة": formulaCells }

];

return AI\_TOOLS\_DIALOGUE.showSummaryTable(stats);

} catch (e) {

HELPERS.logOperation('TOOLS\_SHEETS', 'summarizeSheetStats:error', { sheetName, error: e.message });

return AI\_TOOLS\_DIALOGUE.showError(`❌ فشل تلخيص إحصائيات الورقة "${sheetName}": ${e.message}`);

}

}

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

return {

writeToCell,

clearRange,

insertRowAt,

deleteRow,

filterByValue,

sortColumn,

extractUrls,

highlightEmptyCells,

autoSizeColumns, // ✅ تصدير الدالة الجديدة

renameSheet, // ✅ تصدير الدالة الجديدة

protectRange, // ✅ تصدير الدالة الجديدة

summarizeSheetStats // ✅ تصدير الدالة الجديدة

};

})(HELPERS, AI\_TOOLS\_DIALOGUE, SETTINGS\_PROVIDER, ENV);

/\*\*

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

\* TOOLS\_SHEETS – وثائق الدوال (v15.15)

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

\* يتضمن هذا الملف الدوال التالية:

\*

\* 1. CRUD أساسية:

\* - writeToCell(sheetName, cellA1, value)

\* - clearRange(sheetName, rangeA1)

\* - insertRowAt(sheetName, rowIndex)

\* - deleteRow(sheetName, rowIndex) ← مع تنبيه بريدي

\*

\* 2. التصفية والفرز:

\* - filterByValue(sheetName, columnIndex, value)

\* - sortColumn(sheetName, columnIndex, ascending = true)

\*

\* 3. أدوات تحليل إضافية:

\* - extractUrls(sheetName, columnIndex)

\* - highlightEmptyCells(sheetName, rangeA1, color?)

\*

\* 4. أدوات إدارة الورقة (جديد في v15.15):

\* - autoSizeColumns(sheetName, rangeA1)

\* - renameSheet(oldName, newName)

\* - protectRange(sheetName, rangeA1, editors?)

\* - summarizeSheetStats(sheetName)

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

\*/