Tab 1

{

"timeZone": "Asia/Riyadh",

"dependencies": {

"enabledAdvancedServices": [

{

"userSymbol": "Sheets",

"version": "v4",

"serviceId": "sheets"

},

{

"userSymbol": "Drive",

"version": "v3",

"serviceId": "drive"

},

{

"userSymbol": "Docs",

"version": "v1",

"serviceId": "docs"

},

{

"userSymbol": "DriveActivity",

"version": "v2",

"serviceId": "driveactivity"

},

{

"userSymbol": "DriveLabels",

"version": "v2",

"serviceId": "drivelabels"

},

{

"userSymbol": "Gmail",

"version": "v1",

"serviceId": "gmail"

},

{

"userSymbol": "Calendar",

"version": "v3",

"serviceId": "calendar"

},

{

"userSymbol": "AdminDirectory",

"version": "directory\_v1",

"serviceId": "admin"

},

{

"userSymbol": "WorkspaceEvents",

"version": "v1",

"serviceId": "workspaceevents"

}

]

},

"exceptionLogging": "STACKDRIVER",

"runtimeVersion": "V8",

"webapp": {

"executeAs": "USER\_DEPLOYING",

"access": "ANYONE\_ANONYMOUS"

},

"oauthScopes": [

"https://www.googleapis.com/auth/script.container.ui",

"https://www.googleapis.com/auth/spreadsheets",

"https://www.googleapis.com/auth/drive",

"https://www.googleapis.com/auth/script.send\_mail",

"https://www.googleapis.com/auth/script.scriptapp",

"https://www.googleapis.com/auth/script.external\_request",

"https://www.googleapis.com/auth/script.projects",

"https://www.googleapis.com/auth/userinfo.email"

]

}

/\*\*

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

\* @file 00\_initializer.gs

\* @module GAssistant.Initializer

\* @description

\* هذا الملف هو نقطة الانطلاق وحجر الأساس للمشروع بأكمله.

\*

\* الوظائف الرئيسية:

\* 1. \*\*إنشاء مساحة الاسم:\*\* يقوم بتعريف الكائن العام `GAssistant` مرة واحدة فقط،

\* لتجنب تلوث النطاق العام (Global Scope) وضمان تنظيم الكود.

\* 2. \*\*نقطة الدخول عند الفتح:\*\* يحتوي على دالة `onOpen` التي يتم استدعاؤها

\* تلقائيًا بواسطة Google Sheets، والتي بدورها تشغل التهيئة الأولية لواجهة المستخدم.

\*

\* @architectural\_note يجب أن يكون هذا الملف هو أول ملف يتم تحميله في

\* مشروع Google Apps Script لضمان تهيئة البيئة بشكل صحيح.

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

\*/

// تعريف الكائن العام للمشروع. هذا هو المدخل الوحيد في النطاق العام.

var GAssistant = typeof GAssistant !== "undefined" ? GAssistant : {};

// تهيئة مساحات الأسماء الأساسية لضمان وجودها.

// هذه مجرد تهيئة فارغة لتجنب أخطاء 'undefined' عند محاولة إضافة خصائص لاحقًا.

GAssistant.AI = GAssistant.AI || {};

GAssistant.UI = GAssistant.UI || {};

GAssistant.Tools = GAssistant.Tools || {};

GAssistant.Utils = GAssistant.Utils || {};

GAssistant.Security = GAssistant.Security || {};

GAssistant.Analytics = GAssistant.Analytics || {};

GAssistant.Tests = GAssistant.Tests || {};

GAssistant.Agents = GAssistant.Agents || {};

GAssistant.API = GAssistant.API || {};

// GAssistant.Types لم تعد بحاجة لتهيئة هنا، يتم تعريفها في ملف 03\_types.gs

/\*\*

\* دالة Apps Script القياسية التي تُنفذ عند فتح المستند.

\* هي نقطة الدخول الوحيدة التي تُشغل تهيئة الواجهة.

\* @public

\*/

function onOpen() {

// \*\*التصحيح\*\*: الآن يمكننا استخدام GAssistant.Utils.log بأمان.

// لأن GAssistant.Utils هو IIFE ويتم تنفيذه فورًا عند تحميل الملف 02\_utils.gs.

const Utils = GAssistant.Utils;

Utils.log('G-Assistant: onOpen triggered. Initializing UI...');

// بعد التأكد من أن جميع الوحدات قد تم تحميلها وتهيئة IIFEs الخاصة بها،

// يمكننا استدعاء دالة onOpen الخاصة بوحدة الواجهة GAssistant.UI.

const UI\_Module = GAssistant.UI;

if (UI\_Module && typeof UI\_Module.onOpen === 'function') {

UI\_Module.onOpen();

} else {

// هذا السيناريو يجب ألا يحدث أبداً مع هذا التصميم والترتيب الأبجدي للملفات.

Utils.error("CRITICAL: GAssistant.UI.onOpen is not available. UI initialization failed.");

SpreadsheetApp.getUi().alert("خطأ فادح: فشل تهيئة واجهة المستخدم.");

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --- START OF FILE: 01\_config.gs ---

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

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

\* @file 01\_config.gs

\* @module GAssistant.Config

\* @description

\* الوحدة المركزية لإدارة جميع إعدادات المشروع.

\* تدمج الإعدادات الافتراضية مع الإعدادات المحفوظة من Script Properties

\* وتوفر Caching للأداء العالي. هذا الملف هو "مصدر الحقيقة" الوحيد.

\* \*\*تحديث:\*\* تمت إضافة إعدادات التفكير (Thinking) والتحكم في استدعاء الأدوات.

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

\*/

var GAssistant = GAssistant || {};

// \*\*التصحيح\*\*: تغيير تعريف الوحدة إلى دالة `\_Config\_Module` كما هو مخطط.

// هذه الدالة ستُستدعى في Initializer.

function \_Config\_Module() { // هذا هو المصنع الذي سيعيد وحدة Config

'use strict';

const CACHE = CacheService.getScriptCache();

const CACHE\_KEY = 'g\_assistant\_config\_v6'; // تحديث الإصدار لضمان مسح الكاش القديم

const CACHE\_DURATION = 1800; // 30 دقيقة بالثواني

// مصدر الحقيقة الواحد للإعدادات الافتراضية.

// يمكن للمستخدم تجاوزها عبر ScriptProperties.

const DEFAULTS = {

// General Settings

DEBUG\_MODE: true,

DEFAULT\_LANGUAGE: "ar",

DATE\_FORMAT: "yyyy-MM-dd",

DEFAULT\_CURRENCY: "SAR",

// AI Core Settings

API\_KEY: "", // سيتم جلبه من PropertiesService

GEMINI\_PRO\_MODEL: "gemini-1.5-pro-latest",

GEMINI\_FLASH\_MODEL: "gemini-1.5-flash-latest",

API\_ENDPOINT: "https://generativelanguage.googleapis.com/v1beta/models/",

GENERATION\_CONFIG: {

temperature: 0.7,

topP: 0.95,

maxOutputTokens: 8192,

},

SAFETY\_SETTINGS: [

{ category: "HARM\_CATEGORY\_HARASSMENT", threshold: "BLOCK\_MEDIUM\_AND\_ABOVE" },

{ category: "HARM\_CATEGORY\_HATE\_SPEECH", threshold: "BLOCK\_MEDIUM\_AND\_ABOVE" },

{ category: "HARM\_CATEGORY\_SEXUALLY\_EXPLICIT", threshold: "BLOCK\_MEDIUM\_AND\_ABOVE" },

{ category: "HARM\_CATEGORY\_DANGEROUS\_CONTENT", threshold: "BLOCK\_MEDIUM\_AND\_ABOVE" },

],

// ✅ \*\*الإضافة الجديدة: إعدادات الأدوات والتفكير\*\*

TOOL\_CONFIG: {

functionCallingConfig: {

// "ANY": النموذج يقرر ما إذا كان سيستدعي أداة واحدة أو أكثر.

// "AUTO": النموذج يقرر ما إذا كان سيستدعي أداة. (الافتراضي)

// "NONE": تعطيل استدعاء الأدوات.

mode: "ANY",

},

// تفعيل ميزة التفكير للسماح للنموذج بإظهار خطواته قبل تنفيذ الأداة

thinkingConfig: {

enableThinking: true,

}

},

// System & Logging Settings

OPERATION\_LOG\_SHEET: "سجل العمليات",

DEVELOPMENT\_LOG\_SHEET: "سجل التطوير",

MAX\_LOG\_ENTRIES: 500,

// Tool-specific Settings

REVENUE\_SHEET: "الإيرادات",

EXPENSES\_SHEET: "المصروفات",

// UI Settings

ASSISTANT\_SIDEBAR\_FILE: "ui/AssistantSidebar",

DEVELOPER\_SIDEBAR\_FILE: "ui/DeveloperSidebar",

WEB\_APP\_FILE: "ui/WebApp",

WEB\_APP\_API\_KEY: "", // للحماية عند النشر كتطبيق ويب

PRIMARY\_HEADER\_COLOR: "#4a5568" // لون الرأس الافتراضي

};

let activeConfig = null;

/\*\*

\* يقوم بتحميل الإعدادات عن طريق دمج الإعدادات الافتراضية مع خصائص السكربت.

\* @private

\*/

function \_loadConfig() {

const props = PropertiesService.getScriptProperties().getProperties();

const mergedConfig = { ...DEFAULTS };

for (const key in props) {

// فقط قم بتجاوز الإعدادات المعرفة في DEFAULTS

if (DEFAULTS.hasOwnProperty(key)) {

try {

mergedConfig[key] = JSON.parse(props[key]);

} catch (e) {

mergedConfig[key] = props[key];

}

}

}

if (props.hasOwnProperty('DEBUG\_MODE')) {

const val = String(props.DEBUG\_MODE).toLowerCase();

mergedConfig.DEBUG\_MODE = val === 'true';

}

// اقرأ مفاتيح API الحساسة بشكل منفصل

mergedConfig.API\_KEY = props.GEMINI\_API\_KEY || "";

mergedConfig.WEB\_APP\_API\_KEY = props.WEB\_APP\_API\_KEY || "";

// \*\*التصحيح\*\*: استخدام console.log مباشرة هنا لأنه لا يمكننا الاعتماد على GAssistant.Utils.log في هذه المرحلة المبكرة جدًا من التحميل.

console.log("Config: Configuration loaded from PropertiesService.");

return mergedConfig;

}

/\*\*

\* يجلب كائن الإعدادات الكامل، من الكاش إذا كان متاحًا، أو يقوم بتحميله.

\* @param {boolean} [forceRefresh=false]

\*/

function getAll(forceRefresh = false) {

if (activeConfig && !forceRefresh) return activeConfig;

if (!forceRefresh) {

const cached = CACHE.get(CACHE\_KEY);

if (cached) {

try {

activeConfig = JSON.parse(cached);

return activeConfig;

} catch(e) {

// \*\*التصحيح\*\*: استخدام console.warn مباشرة هنا.

console.warn("Config: Could not parse cached config. Reloading.");

}

}

}

activeConfig = \_loadConfig();

CACHE.put(CACHE\_KEY, JSON.stringify(activeConfig), CACHE\_DURATION);

return activeConfig;

}

/\*\*

\* يجلب قيمة إعداد معين.

\* @param {string} key

\*/

function get(key) {

return getAll()[key];

}

/\*\*

\* دالة للتحقق من صحة الإعدادات الأساسية.

\* @returns {boolean}

\*/

function validate() {

// \*\*التصحيح\*\*: هذه الدالة تعتمد على GAssistant.Utils.

// لذا، يجب أن تُستدعى فقط بعد تهيئة GAssistant.Utils.

// سنقوم بجلبها داخل الدالة لضمان توفرها.

const { Utils } = GAssistant.Utils.Injector.get('Utils');

const config = getAll(true); // فرض التحديث لضمان أحدث الإعدادات

const requiredKeys = ['API\_KEY', 'API\_ENDPOINT', 'GEMINI\_PRO\_MODEL'];

for (const key of requiredKeys) {

if (!config[key] || (typeof config[key] === 'string' && config[key].trim() === '')) {

Utils.error(`Config Validation FAILED: Required key "${key}" is missing or empty.`); // استخدام Utils.error

return false;

}

}

Utils.log("Config Validation PASSED: Essential configuration keys are present."); // استخدام Utils.log

return true;

}

// \*\*التصحيح\*\*: استخدام defineModule هنا.

// Config لا تعتمد على وحدات GAssistant أخرى في تعريفها الأولي.

const defineModule = GAssistant.Utils.defineModule;

return defineModule([], () => {

return {

get,

getAll,

validate,

};

}, 'Config'); // Label للوحدة

} // نهاية تعريف الدالة \_Config\_Module

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --- START OF FILE: 02\_utils.gs ---

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

\* @file 02\_utils.gs

\* @module GAssistant.Utils

\* @description

\* وحدة الأدوات المساعدة العامة والموحدة مع نظام حقن التبعيات وفحص البيئة.

\* تشمل أدوات تسجيل، فحص، تحليل، وتنفيذ آمن لدوال النظام.

\*/

// تأكيد وجود الكائن الرئيسي قبل أي تنفيذ

var GAssistant = typeof GAssistant !== 'undefined' ? GAssistant : {};

GAssistant.Utils = GAssistant.Utils || {};

// تعريف الوحدة الفعلية كـ IIFE تُنفّذ فورًا عند تحميل السكريبت

GAssistant.Utils = (() => {

'use strict';

// ==================== تحميل التكوين (بشكل آمن) ====================

// يتم الوصول إلى GAssistant.Config هنا بشكل آمن، بعد أن يكون قد تم تعريفه كـ IIFE.

function \_getConfig() {

return (GAssistant.Config && typeof GAssistant.Config.getAll === 'function') ? GAssistant.Config.getAll() : {

DEBUG\_MODE: true,

OPERATION\_LOG\_SHEET: "ErrorLog",

MAX\_LOG\_ENTRIES: 500,

PRIMARY\_HEADER\_COLOR: "#4a5568" // قيمة افتراضية لتجنب الخطأ

};

}

// ==================== أدوات التسجيل ====================

function log(message, data) {

console.log(`[INFO] ${message}`, data ? JSON.stringify(data) : '');

if (\_getConfig().DEBUG\_MODE) \_logToSheet('INFO', message, data);

}

function warn(message, data) {

console.warn(`[WARN] ${message}`, data ? JSON.stringify(data) : '');

if (\_getConfig().DEBUG\_MODE) \_logToSheet('WARN', message, data);

}

function error(message, err) {

const details = err instanceof Error ? { message: err.message, stack: err.stack } : err;

console.error(`[ERROR] ${message}`, JSON.stringify(details));

\_logToSheet('ERROR', message, details);

}

function \_logToSheet(level, message, data) {

try {

const config = \_getConfig();

// تأكد أن getSheet و GAssistant.Config متاحتان

if (typeof getSheet !== 'function' || !config.OPERATION\_LOG\_SHEET) {

console.error(`Utils: Cannot log to sheet - getSheet or config missing.`);

return;

}

const sheet = getSheet(config.OPERATION\_LOG\_SHEET, ["Timestamp", "Level", "Message", "Data"]);

if (!sheet) return;

const compact = data ? JSON.stringify(data).substring(0, 1000) : '';

sheet.appendRow([new Date(), level, message, compact]);

const maxRows = config.MAX\_LOG\_ENTRIES || 500;

if (sheet.getLastRow() > maxRows) {

sheet.deleteRows(2, sheet.getLastRow() - maxRows);

}

} catch (e) {

console.error(`Utils: Failed to log to sheet - ${e.message}`);

}

}

// ==================== فحص الورقة ====================

function getSheet(name, headers) {

if (!name || typeof name !== 'string') {

error('getSheet: اسم الورقة غير صالح.');

return null;

}

const ss = SpreadsheetApp.getActiveSpreadsheet();

let sheet = ss.getSheetByName(name);

if (sheet) {

const isHidden = sheet.isSheetHidden?.() ?? false;

// ✅ تصحيح: فحص وجود getProtections قبل الاستدعاء

const isProtected = sheet.getProtections ? (sheet.getProtections(SpreadsheetApp.ProtectionType.SHEET)?.length > 0) : false;

if (isHidden || isProtected) {

warn(`getSheet: الورقة "${name}" مخفية أو محمية.`);

return null;

}

} else {

sheet = ss.insertSheet(name);

if (Array.isArray(headers) && headers.length > 0) {

const range = sheet.getRange(1, 1, 1, headers.length);

const headerColor = \_getConfig().PRIMARY\_HEADER\_COLOR;

range.setValues([headers])

.setFontWeight('bold')

.setBackground(headerColor)

.setFontColor('#ffffff');

}

}

return sheet;

}

// ==================== تنفيذ آمن ====================

function executeSafely(func, args = [], context = '') {

const start = Date.now();

try {

const result = func(...args);

log(`Execution Time: '${func.name || context}'`, { duration\_ms: Date.now() - start });

return result;

} catch (e) {

const duration = Date.now() - start;

error(`Execution failed: ${func.name || 'anonymous'} ${context}`, e);

log(`Execution Time (Failed): '${func.name || context}'`, { duration\_ms: duration });

const Dialogue = GAssistant?.UI?.Dialogue; // الوصول الآمن لوحدة الحوار

return Dialogue ? Dialogue.createError(`حدث خطأ: ${e.message}`) : { type: 'error', text: e.message };

}

}

// ==================== أدوات إضافية ====================

function getFunctionByPath(path) {

// هذا يسمح بالوصول إلى وحدات متداخلة مثل GAssistant.AI.Core

// يعتمد على أن GAssistant تم تهيئته بالكامل في النطاق العام.

return path.split('.').reduce((obj, key) => obj?.[key], GAssistant);

}

function safeParse(str) {

try { return JSON.parse(str); } catch (\_) { return null; }

}

// ==================== حقن التبعيات ====================

const Injector = {

get: (...moduleNames) => {

const result = {};

for (const fullName of moduleNames) {

// الوصول للوحدات عبر GAssistant بعد أن تكون قد تم تعريفها في النطاق العام.

const mod = getFunctionByPath(`GAssistant.${fullName}`);

if (typeof mod === 'undefined' || mod === null) {

const msg = `CRITICAL DEPENDENCY ERROR: Module GAssistant.${fullName} is not loaded or does not exist.`;

// نستخدم error() الخاص بـ Utils نفسه هنا

error(msg);

throw new Error(msg);

}

// لا يوجد دعم لـ Lazy Initializers هنا، لأن كل الوحدات IIFE وتُنفذ فورًا

const key = fullName.split('.').pop();

result[key] = mod;

}

return result;

}

};

// ==================== دالة تعريف الوحدة ====================

/\*\*

\* دالة مساعدة لتعريف الوحدات. تُستخدم لتوحيد عملية حقن التبعيات وتسجيل الوحدة.

\* @param {string[]} deps - مصفوفة بمسارات التبعيات المطلوبة (e.g., ['Utils', 'UI.Dialogue']).

\* @param {function(object): any} factory - دالة المصنع التي تنفذ منطق الوحدة وتستقبل التبعيات.

\* @param {string} label - تسمية الوحدة للتسجيل (e.g., 'AI.Core').

\* @returns {any} - الناتج الذي تُرجعه دالة المصنع (الوحدة نفسها).

\*/

function defineModule(deps, factory, label) {

// استخدام Injector لجلب التبعيات

const resolvedDeps = Injector.get(...deps);

// تنفيذ دالة المصنع مع التبعيات المحقونة

const output = factory(resolvedDeps);

// تسجيل الوحدة عند تهيئتها بنجاح

if (label) {

log(`📦 Module Ready: ${label}`);

}

return output;

}

// ==================== أدوات فحص هيكل التحميل (للتصحيح) ====================

/\*\*

\* يتحقق من وجود كل الوحدات الحيوية داخل GAssistant (لأغراض التصحيح).

\* ويسجل حالة كل واحدة (موجود / مفقودة).

\* @returns {void}

\*/

function verifyCoreModules() {

// لا يمكننا الاعتماد على GAssistant.Utils هنا، لأن هذه هي الوحدة نفسها.

// نستخدم log, warn, error من النطاق المحلي.

const requiredModules = [

'AI', 'AI.Core', 'AI.Memory', 'Agents', 'Tools', 'Tools.Catalog',

'UI', 'UI.Dialogue', 'Config', 'Utils', 'Security', 'Analytics',

'Tests', 'API', 'Types' // هذا ليس موديول JavaScript عاديًا، هو فقط لـ JSDoc

];

log("🔍 بدء فحص وحدات GAssistant الأساسية...", { total: requiredModules.length });

requiredModules.forEach((path) => {

let module = getFunctionByPath(`GAssistant.${path}`);

if (!module) {

error(`❌ الوحدة "GAssistant.${path}" غير محملة`);

} else {

log(`✅ الوحدة "GAssistant.${path}" محملة`);

// هذا الجزء يخص Types فقط، يجب استثناؤه من الفحص العام للوحدات

if (path === 'Types') {

if (typeof GAssistant.Types !== 'undefined') {

log(`✅ الوحدة "GAssistant.Types" (JSDoc) موجود`);

} else {

warn(`⚠️ الوحدة "GAssistant.Types" (JSDoc) غير موجودة.`);

}

}

}

});

log("✅ تم الانتهاء من فحص الوحدات.");

}

// ==================== تصدير ====================

return {

log, warn, error,

getSheet, executeSafely,

getFunctionByPath, safeParse,

Injector,

defineModule,

verifyCoreModules // ✅ تصدير الدالة الجديدة لأغراض التصحيح

};

})();

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --- END OF FILE: 02\_utils.gs ---

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --- START OF FILE: 03\_types.gs ---

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*

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

\* @file 03\_types.gs

\* @description

\* هذا الملف هو المستودع المركزي لتعريفات الأنواع (Type Definitions) باستخدام

\* JSDoc. لا يحتوي على أي كود تنفيذي، بل هو بمثابة "قاموس بيانات" للمشروع

\* لتحسين وضوح الكود وتفعيل ميزات المساعدة في محرر الأكواد (IntelliSense).

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

\*/

// \*\*التصحيح\*\*: إضافة تعريف GAssistant.Types لتوحيد نمط التعريف، حتى لو لم يكن يحتوي على كود تنفيذي.

var GAssistant = GAssistant || {};

GAssistant.Types = GAssistant.Types || {};

// لا يوجد IIFE هنا، هذا الملف لا يقوم بأي تنفيذ، فقط تعريفات JSDoc.

/\*\*

\* يمثل الحالة الكاملة لطلب واحد يتم معالجته بواسطة محرك الذكاء الاصطناعي.

\* يتم إنشاء هذا الكائن وتمريره بين وحدات AI المختلفة.

\* @typedef {Object} AiState

\* @property {string} userPrompt - طلب المستخدم الأصلي.

\* @property {object} options - خيارات إضافية للطلب (مثل modelOverride, structuredOutputRequested).

\* @property {string | null} model - اسم نموذج Gemini الذي تم اختياره للمهمة.

\* @property {string | null} systemInstruction - التعليمات المجمعة للنظام التي تم بناؤها بواسطة GAssistant.AI.Context.

\* @property {Array<object>} history - سجل المحادثة من الجلسة الحالية.

\* @property {Array<object>} tools - تعريفات الأدوات (Function Declarations) المتاحة للنموذج.

\* @property {UiResponse | null} final - النتيجة النهائية المنسقة التي ستُرسل للواجهة الأمامية.

\*/

/\*\*

\* يمثل كائن الرد الموحد الذي يتم إرساله دائمًا إلى الواجهة الأمامية (HTML).

\* هذا يضمن أن جافا سكريبت في الواجهة تتعامل دائمًا مع هيكل بيانات متوقع.

\* @typedef {Object} UiResponse

\* @property {'info'|'success'|'warning'|'error'|'table'|'suggestions'} type - نوع الرسالة لتحديد كيفية عرضها.

\* @property {string} text - النص الرئيسي للرسالة أو عنوان الجدول.

\* @property {any} [data] - بيانات إضافية اختيارية. في حالة الجداول، يكون هذا الكائن { headers: string[], rows: any[][] }.

\*/

/\*\*

\* يمثل تعريف أداة واحدة كما هي مسجلة في كتالوج الأدوات.

\* @typedef {Object} ToolDefinition

\* @property {string} name - اسم الدالة الذي سيستخدمه Gemini لاستدعائها.

\* @property {string} description - وصف واضح وموجز لوظيفة الأداة.

\* @property {object} parameters - مخطط JSON للمعاملات التي تتوقعها الدالة.

\* @property {string} functionPath - المسار الكامل للدالة داخل كائن GAssistant لتسهيل استدعائها برمجيًا.

\*/

/\*\*

\* يمثل نتيجة اختبار واحدة كما يتم إرجاعها من وحدة الاختبارات.

\* @typedef {Object} TestResult

\* @property {string} name - اسم الاختبار الذي تم تشغيله.

\* @property {boolean} success - هل نجح الاختبار (true) أم فشل (false).

\* @property {string} message - رسالة توضيحية للنتيجة (مثل "نجح" أو رسالة الخطأ).

\* @property {number} duration - مدة تنفيذ الاختبار بالمللي ثانية.

\*/

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// --- END OF FILE: 03\_types.gs ---

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tab 2