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

// --- 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\*\*: تعريف الوحدة داخل دالة عامة. هذه الدالة سيتم استدعاؤها في \_initializeGAssistantSystem().

function \_Config\_Module() { // لم يعد IIFE يتم تنفيذه فورًا هنا.

'use strict';

const CACHE = CacheService.getScriptCache();

const CACHE\_KEY = 'g\_assistant\_config\_v8'; // تحديث الإصدار لضمان مسح الكاش القديم بعد التغيير في الهيكلة

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];

}

/\*\*

\* ✅ \*\*التحسين الذكي\*\*: دالة للتحقق من وجود مفتاح إعداد معين.

\* تسمح بالتحقق من وجود مفتاح قبل محاولة استدعائه.

\* @param {string} key - المفتاح المراد التحقق من وجوده.

\* @returns {boolean} - true إذا كان المفتاح موجوداً في الإعدادات، وإلا false.

\*/

function has(key) {

return Object.prototype.hasOwnProperty.call(getAll(), key);

}

/\*\*

\* دالة للتحقق من صحة الإعدادات الأساسية.

\* تُستخدم للتأكد من وجود المفاتيح الإلزامية.

\* @returns {boolean}

\*/

function validate() {

// \*\*التصحيح\*\*: هذه الدالة تعتمد على GAssistant.Utils.Injector.get، لذا يجب أن تُستدعى فقط بعد تهيئة GAssistant.Utils.

// عادةً ما تُستدعى هذه الدالة في onOpen بعد تهيئة جميع الوحدات الأساسية.

const { Utils } = GAssistant.Utils.Injector.get('Utils'); // الوصول إلى Utils بعد أن تكون متاحة

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

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

for (const key of requiredKeys) {

if (!has(key) || (typeof config[key] === 'string' && config[key].trim() === '')) { // استخدام دالة has الجديدة

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

Logger.log(`❌ Config Validation FAILED: Required key "${key}" is missing or empty.`); // تسجيل في Logger أيضًا

return false;

}

}

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

Logger.log("✅ Config Validation PASSED: Essential configuration keys are present."); // تسجيل في Logger أيضًا

return true;

}

/\*\*

\* ✅ \*\*التحسين الذكي\*\*: دالة لتصدير جميع الإعدادات الحالية إلى ورقة عمل لأغراض التصحيح أو الإدارة.

\* @param {string} [sheetName="Config Dump"] - اسم الورقة التي سيتم تصدير الإعدادات إليها.

\*/

function dump(sheetName = "Config Dump") {

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

return Utils.executeSafely(() => {

const config = getAll();

const sheet = Utils.getSheet(sheetName, ["Key", "Value", "Type"]); // استخدام Utils.getSheet

// مسح المحتوى القديم للحفاظ على النظافة

if (sheet.getLastRow() > 1) {

sheet.getRange(2, 1, sheet.getLastRow() - 1, sheet.getLastColumn()).clearContent();

}

for (const key in config) {

if (config.hasOwnProperty(key)) {

let value = config[key];

let type = typeof value;

if (type === 'object') {

value = JSON.stringify(value, null, 2); // تنسيق الكائنات كـ JSON

}

sheet.appendRow([key, value, type]);

}

}

Utils.log(`Config: Settings dumped to sheet "${sheetName}".`);

SpreadsheetApp.getUi().alert(`✅ تم تصدير الإعدادات إلى ورقة "${sheetName}".`);

}, [], `dumping config to sheet ${sheetName}`);

}

return {

get,

getAll,

has, // تصدير الدالة الجديدة

validate,

dump, // تصدير الدالة الجديدة

};

} // نهاية تعريف الدالة `\_Config\_Module`