/\*\*

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

\* @file ai\_ask.gs – v15.7 (وحدة الاتصال بالـ API)

\* @module AI\_ASK

\* @description

\* هذه الوحدة هي "ساعي البريد" المسؤول عن الاتصال بنموذج Gemini:

\* • عزل كامل لمنطق الاتصال الخارجي (UrlFetchApp)

\* • آلية إعادة محاولة تلقائية (Retry Mechanism) مع تراجع أسي (Exponential Backoff)

\* • بناء حمولة (Payload) متوافقة مع Function Calling API ودعم الوسائط المتعددة (Multi-modal)

\* • التحقق من صحة النموذج عبر ALLOWED\_MODELS

\* • دمج إعدادات التفكير (Thinking) من AI\_CONFIG

\* • توحيد عرض رسائل الخطأ النهائية باستخدام AI\_TOOLS\_DIALOGUE

\* • دعم كامل لـ generationConfig, safetySettings, و toolConfig (بما في ذلك thinkingConfig)

\* • دعم الإخراج المهيكل (structuredOutput) ودعم الصور (inlineData)

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

\*/

/\*\*

\* @typedef {object} GeminiResponsePayload

\* @property {Array<object>} candidates - قائمة بالاستجابات المحتملة من النموذج.

\* @property {object} [usageMetadata] - بيانات استخدام الرموز المميزة.

\*/

/\*\*

\* @typedef {object} APIResponse

\* @property {'success'|'error'} type - نوع الاستجابة.

\* @property {GeminiResponsePayload|string} payload - الحمولة الفعلية للاستجابة أو رسالة الخطأ.

\* @property {number} statusCode - رمز حالة HTTP للاستجابة.

\*/

/\*\*

\* @typedef {object} PipelineState

\* @property {GoogleAppsScript.Spreadsheet.Spreadsheet} ss - كائن جدول البيانات النشط.

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

\* @property {string} model - اسم النموذج الذي سيتم استخدامه (مثلاً 'gemini-1.5-pro').

\* @property {string} role - دور المستخدم.

\* @property {object} ctx - سياق الجلسة المحفوظ من AI\_MEMORY.

\* @property {boolean} halt - علامة لإيقاف الـ pipeline مبكراً.

\* @property {object|null} final - النتيجة النهائية التي سيتم إرجاعها من AI\_CORE.

\* @property {APIResponse|null} geminiResponse - الاستجابة الخام من نموذج Gemini بعد التحليل الأولي.

\* @property {string} systemInstruction - تعليمات النظام التي سيتم إرسالها للنموذج.

\* @property {string|null} [imagePart] - بيانات الصورة بصيغة Base64 (اختياري، للمدخلات متعددة الوسائط).

\* @property {boolean} [structuredOutputRequested=false] - هل طلب المستخدم إخراجًا مهيكلًا (JSON)؟

\*/

var AI\_ASK = (function(AI\_CONFIG, HELPERS, TOOLS\_CATALOG, AI\_TOOLS\_DIALOGUE) {

'use strict';

// جلب جميع الإعدادات من AI\_CONFIG لتوحيد المصدر

const config = AI\_CONFIG.getConfig();

/\*\*

\* @function getFunctionDeclarations

\* @description يجلب تعريفات الأدوات للـ Function Calling (JSON).

\* @returns {Array<object>}

\*/

function getFunctionDeclarations() {

return TOOLS\_CATALOG.getFunctionList();

}

/\*\*

\* @function getEndpoint

\* @description يركب رابط API لنموذج محدد ومفتاح API.

\* @param {string} model - اسم النموذج.

\* @param {string} apiKey - مفتاح API.

\* @returns {string} - رابط نقطة نهاية API.

\*/

function getEndpoint(model, apiKey) {

return `https://generativelanguage.googleapis.com/v1beta/models/` +

`${encodeURIComponent(model)}:generateContent?key=${apiKey}`;

}

/\*\*

\* @function callGeminiAgent

\* @description

\* تنفّذ اتصالاً بنموذج Gemini مع retries وتطبق آلية التراجع الأسي.

\* تُحدّث state.geminiResponse أو state.final + state.halt بناءً على نتيجة الاتصال.

\* تدعم الآن التفكير الداخلي، الإخراج المهيكل، والمدخلات متعددة الوسائط (الصور).

\* @param {PipelineState} state – حالة الـ Pipeline من AI\_CORE.

\* @returns {PipelineState} state المحدث.

\*/

function callGeminiAgent(state) {

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:start', { model: state.model });

if (state.halt) {

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:pre\_halted', { model: state.model });

return state;

}

// التحقق من صحة النموذج باستخدام ALLOWED\_MODELS من AI\_CONFIG

if (!config.ALLOWED\_MODELS.includes(state.model)) {

state.final = AI\_TOOLS\_DIALOGUE.showError(`🚫 النموذج "${state.model}" غير مسموح به حسب الإعدادات.`);

state.halt = true;

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:end', { status: 'failed', reason: 'invalid\_model', model: state.model });

return state;

}

const apiKey = HELPERS.getGeminiApiKey();

if (!apiKey) {

state.final = AI\_TOOLS\_DIALOGUE.showError('🔐 مفتاح API غير موجود.');

state.halt = true;

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:end', { status: 'failed', reason: 'no\_api\_key' });

return state;

}

const endpoint = getEndpoint(state.model, apiKey);

// 1. بناء `contents` (الرسائل)

const contents = [

{

role: 'user',

parts: [{ text: state.prompt }]

}

];

// إضافة تعليمات النظام كجزء من المحتوى (عادةً في بداية أول رسالة للمستخدم)

if (state.systemInstruction) {

contents[0].parts.unshift({ text: state.systemInstruction });

}

// دعم الصور (inlineData)

if (state.imagePart) {

contents[0].parts.push({

inlineData: {

mimeType: 'image/png', // يمكن جعل هذا ديناميكيًا إذا كانت هناك أنواع صور مختلفة

data: state.imagePart

}

});

}

// 2. إعدادات الجيل (generationConfig)

const generationConfig = {

temperature: 0.7, // يمكن جعلها قابلة للتكوين في ENV.SETTINGS

maxOutputTokens: 2048 // يمكن جعلها قابلة للتكوين في ENV.SETTINGS

};

// دعم الإخراج المهيكل (structuredOutput)

if (state.structuredOutputRequested) {

generationConfig.responseMimeType = "application/json";

}

// 3. إعدادات الأمان (safetySettings)

// يمكن جعل هذه الإعدادات قابلة للتكوين في ENV.SETTINGS

const safetySettings = [

{ category: 'HARM\_CATEGORY\_HARASSMENT', threshold: 'BLOCK\_NONE' },

{ category: 'HARM\_CATEGORY\_HATE\_SPEECH', threshold: 'BLOCK\_NONE' },

{ category: 'HARM\_CATEGORY\_SEXUALLY\_EXPLICIT', threshold: 'BLOCK\_NONE' },

{ category: 'HARM\_CATEGORY\_DANGEROUS\_CONTENT', threshold: 'BLOCK\_NONE' }

];

// 4. إعدادات الأدوات والتفكير (toolConfig)

const toolConfig = {

functionCallingConfig: {

mode: 'AUTO', // أو 'ANY' أو 'NONE'

allowedFunctionNames: TOOLS\_CATALOG.getFunctionList().map(f => f.name) // قائمة بأسماء الدوال المسموح بها

}

};

// إضافة thinkingConfig إذا كان مفعلاً من AI\_CONFIG

if (config.INCLUDE\_THOUGHTS) {

toolConfig.thinkingConfig = {

budget: config.THINKING\_BUDGET

};

}

// بناء حمولة الطلب النهائية

const requestPayload = {

contents: contents,

generationConfig: generationConfig,

safetySettings: safetySettings, // ✅ إضافة safetySettings

tools: [{ functionDeclarations: getFunctionDeclarations() }], // تعريفات الأدوات

toolConfig: toolConfig // إعدادات الأدوات والتفكير

};

const options = {

method: 'post',

contentType: 'application/json',

payload: JSON.stringify(requestPayload),

muteHttpExceptions: true

};

for (let attempt = 1; attempt <= config.MAX\_API\_RETRIES; attempt++) {

try {

const start = Date.now();

const res = UrlFetchApp.fetch(endpoint, options);

const latency = Date.now() - start;

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:metrics', { attempt, latency, model: state.model });

const code = res.getResponseCode();

const body = res.getContentText();

if (code === 200) {

const json = JSON.parse(body);

state.geminiResponse = /\*\* @type {APIResponse} \*/ ({

type: 'success',

payload: json,

statusCode: 200

});

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:end', { status: 'success', attempt, model: state.model });

return state;

} else {

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:http\_error', { code, attempt, body, model: state.model });

// إذا كان الخطأ 4xx (عميل)، قد لا يكون إعادة المحاولة مفيداً

if (code >= 400 && code < 500) {

state.final = AI\_TOOLS\_DIALOGUE.showError(`⚠️ حدث خطأ في الطلب (رمز ${code}): ${body.substring(0, 200)}`);

state.halt = true;

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:end', { status: 'failed\_client\_error', code, model: state.model });

return state;

}

}

} catch (e) {

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:exception', { message: e.message, attempt, model: state.model });

}

// تطبيق التراجع الأسي (Exponential Backoff)

if (attempt < config.MAX\_API\_RETRIES) {

const delay = config.RETRY\_DELAY\_MS \* Math.pow(2, attempt - 1);

Utilities.sleep(delay);

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:retrying', { attempt, delay });

}

}

state.final = AI\_TOOLS\_DIALOGUE.showError(`💥 فشل الاتصال بنموذج Gemini بعد ${config.MAX\_API\_RETRIES} محاولات.`);

state.halt = true;

HELPERS.logOperation('AI\_ASK', 'callGeminiAgent:end', { status: 'failed\_max\_retries', model: state.model });

return state;

}

return {

callGeminiAgent

};

})(AI\_CONFIG, HELPERS, TOOLS\_CATALOG, AI\_TOOLS\_DIALOGUE);