/\*\*

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

\* @file ai\_core.gs

\* @module GAssistant.AI.Core

\* @description

\* المحرك الرئيسي لمعالجة طلبات الذكاء الاصطناعي.

\* يعمل كمنظم أوركسترا يدير دورة حياة الطلب من البداية إلى النهاية.

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

\*/

var GAssistant = GAssistant || {};

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

GAssistant.AI.Core = (() => {

'use strict';

// --- تعريف التبعيات الرئيسية ---

// ملاحظة: لا نستخدم تفكيك (destructuring) للوحدات التي قد لا تكون محملة بعد.

// سنشير إليها مباشرة عند الحاجة.

const Utils = GAssistant.Utils;

const Config = GAssistant.Config;

const Memory = GAssistant.AI.Memory;

const Context = GAssistant.AI.Context;

const Dispatcher = GAssistant.AI.Dispatcher;

/\*\*

\* الدالة الرئيسية للتفاعل مع نموذج Gemini.

\* تدير دورة حياة الطلب: بناء السياق، استدعاء API، ومعالجة الرد.

\* @param {string} userPrompt - طلب المستخدم.

\* @param {object} [options={}] - خيارات إضافية مثل { modelOverride, structuredOutputRequested }.

\* @returns {object} - كائن الرد النهائي المنسق للواجهة الأمامية.

\*/

function ask(userPrompt, options = {}) {

const startTime = Date.now();

Utils.log('AI.Core.ask: Start', { prompt: userPrompt.substring(0, 70), options });

const state = {

userPrompt,

options,

model: null,

systemInstruction: null,

history: [],

tools: [],

final: null,

};

try {

// 1. تحديد النموذج المناسب للمهمة

const config = Config.getAll();

state.model = options.modelOverride || (userPrompt.length > 500 ? config.GEMINI\_PRO\_MODEL : config.GEMINI\_FLASH\_MODEL);

Utils.log(`AI.Core.ask: Model selected -> ${state.model}`);

// 2. بناء السياق الكامل (System Instruction, History, Tools)

Context.build(state);

// 3. بناء حمولة الطلب النهائية لـ Gemini API

const payload = \_buildPayload(state);

// 4. استدعاء Gemini API

const apiResponse = \_callGeminiAPI(state.model, payload);

// 5. معالجة الرد وتوجيهه (نص، استدعاء أداة، خطأ)

state.final = Dispatcher.processApiResponse(apiResponse);

// 6. تحديث ذاكرة الجلسة بالطلب والرد النهائي

Memory.addMessageToHistory('user', userPrompt);

Memory.addMessageToHistory('model', state.final);

} catch (e) {

Utils.error('AI.Core.ask: Unhandled exception', e);

// \*\*التصحيح الرئيسي هنا\*\*

// استدعاء الدالة مباشرة من مساحة الاسم الكاملة بدلاً من استخدام المتغير المفكك.

state.final = GAssistant.UI.Dialogue.createError(`💥 حدث خطأ كارثي في محرك الذكاء الاصطناعي: ${e.message}`);

}

const duration = Date.now() - startTime;

Utils.log(`AI.Core.ask: End. Total duration: ${duration}ms`, { finalType: state.final?.type || 'unknown' });

return state.final;

}

/\*\*

\* دالة مساعدة داخلية لبناء حمولة الطلب النهائية.

\* @private

\*/

function \_buildPayload(state) {

const config = Config.getAll();

const payload = {

contents: [...state.history, { role: "user", parts: [{ text: state.userPrompt }] }],

system: { parts: [{ text: state.systemInstruction }] }, // استخدام system بدلاً من systemInstruction

tools: state.tools,

generationConfig: { ...config.GENERATION\_CONFIG }

};

if (state.options.structuredOutputRequested) {

payload.generationConfig.response\_mime\_type = "application/json"; // المفتاح الصحيح هو response\_mime\_type

}

return payload;

}

/\*\*

\* دالة مساعدة داخلية لاستدعاء Gemini API.

\* @private

\*/

function \_callGeminiAPI(model, payload) {

const config = Config.getAll();

const apiKey = config.API\_KEY;

if (!apiKey) {

throw new Error("مفتاح Gemini API غير موجود. يرجى إعداده في خصائص المشروع.");

}

const url = `${config.API\_ENDPOINT}${model}:generateContent?key=${apiKey}`;

const options = {

method: 'post',

contentType: 'application/json',

payload: JSON.stringify(payload),

muteHttpExceptions: true

};

Utils.log('AI.Core.\_callGeminiAPI: Sending request', { url, payloadSize: options.payload.length });

const response = UrlFetchApp.fetch(url, options);

const responseCode = response.getResponseCode();

const responseBody = response.getContentText();

if (responseCode === 200) {

Utils.log('AI.Core.\_callGeminiAPI: Received successful response');

return JSON.parse(responseBody);

} else {

Utils.error('AI.Core.\_callGeminiAPI: API Error', { responseCode, responseBody });

throw new Error(`Gemini API Error (${responseCode}): ${responseBody}`);

}

}

return {

ask

};

})();