/\*\*

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

\* @file tools\_dialogue.gs – v15.7

\* @module AI\_TOOLS\_DIALOGUE

\* @description

\* أدوات التفاعل والذاكرة والسياق للمساعد الذكي G-Assistant.

\* تشمل:

\* - توحيد هيكل الردود (response format)

\* - دعم سياق الجلسة مع صلاحية مؤقتة (expiresAt)

\* - توصيات ذكية مثل suggestModelSwitch و suggestNextActions

\* - تسجيل كل دالة في سجل الاستخدام (عبر HELPERS.logOperation)

\* تعتمد على: ENV, AI\_MEMORY, HELPERS, AI\_CORE

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

\*/

var AI\_TOOLS\_DIALOGUE = (function(ENV, AI\_MEMORY, HELPERS, AI\_CORE) {

'use strict';

const SETTINGS = ENV.SETTINGS;

// ========== 1. رسائل التفاعل ==========

/\*\*

\* @function showMessage

\* @description

\* تنشئ كائن رسالة موحد لعرضه في واجهة المستخدم.

\* @param {string} level - مستوى الرسالة (success, warning, error, info).

\* @param {string} text - نص الرسالة.

\* @returns {object} - كائن الرسالة.

\*/

function showMessage(level, text) {

const emojiMap = { error: "🚨", warning: "⚠️", success: "✅", info: "ℹ️" };

return { type: "message", level, text: `${emojiMap[level] || '💬'} ${text}` };

}

/\*\*

\* @function showError

\* @description

\* تعرض رسالة خطأ وتسجلها.

\* @param {string} errorMessage - رسالة الخطأ.

\* @returns {object} - كائن رسالة الخطأ.

\*/

function showError(errorMessage) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'showError', { error: errorMessage });

return showMessage('error', errorMessage);

}

/\*\*

\* @function askUserConfirmation

\* @description

\* يطلب تأكيد المستخدم قبل تنفيذ عملية حساسة.

\* @param {string} message - الرسالة التي تظهر للمستخدم لطلب التأكيد.

\* @param {string} [confirmationType='normal'] - نوع التأكيد (مثلاً 'normal', 'critical').

\* @param {object} [payload={}] - بيانات إضافية يمكن تمريرها مع طلب التأكيد.

\* @returns {object} - كائن يمثل طلب التأكيد.

\*/

function askUserConfirmation(message, confirmationType = 'normal', payload = {}) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'askUserConfirmation', { message, type: confirmationType });

return {

type: 'confirmation\_required',

action: '\_internal\_userConfirmed', // هذه الدالة يجب أن تكون في tools\_internal.gs

params: { payload, confirmationType },

message

};

}

/\*\*

\* @function showSummaryTable

\* @description

\* يعرض بيانات JSON في شكل جدول نصي منسق.

\* @param {string|object} jsonData - البيانات المراد عرضها، يمكن أن تكون سلسلة JSON أو كائن/مصفوفة.

\* @returns {object} - كائن يحتوي على الجدول المنسق أو رسالة خطأ.

\*/

function showSummaryTable(jsonData) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'showSummaryTable', {});

try {

const data = typeof jsonData === 'string' ? JSON.parse(jsonData) : jsonData;

if (!Array.isArray(data) || data.length === 0) {

return showMessage("info", "لا توجد بيانات لعرضها في الجدول.");

}

const headers = Object.keys(data[0]);

// ✅ تحسين: التأكد من تحويل h إلى String قبل حساب الطول

const colWidths = headers.map(h => String(h).length);

data.forEach(row => {

headers.forEach((h, i) => {

const len = String(row[h] ?? '').length;

if (len > colWidths[i]) colWidths[i] = len;

});

});

// ✅ تحسين: التأكد من تحويل h إلى String قبل استخدام padEnd

const headerLine = headers.map((h, i) => String(h).padEnd(colWidths[i])).join(" │ ");

const divider = colWidths.map(w => "─".repeat(w)).join("─┼─");

const bodyRows = data.map(row =>

headers.map((h, i) => String(row[h] ?? '').padEnd(colWidths[i])).join(" │ ")

);

return {

type: 'summary',

table: "```\n" + headerLine + "\n" + divider + "\n" + bodyRows.join("\n") + "\n```"

};

} catch (e) {

return showError(`فشل عرض الجدول: ${e.message}`);

}

}

// ========== 2. إدارة السياق اللحظي ==========

/\*\*

\* @function setContext

\* @description

\* يحفظ قيمة في سياق الجلسة. يتم منع حفظ المفاتيح الحساسة.

\* @param {string} key - المفتاح الذي سيتم حفظ القيمة تحته.

\* @param {\*} value - القيمة المراد حفظها (سيتم تحويل الكائنات إلى JSON).

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

\*/

function setContext(key, value) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'setContext:start', { key }); // ✅ تحسين: سجل بداية العملية

const forbidden = ["token", "sessionid", "apikey", "password"];

if (forbidden.includes(key.toLowerCase())) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'setContext:forbidden\_key', { key }); // ✅ تحسين: سجل سبب الرفض

return showMessage("warning", `⚠️ المفتاح "${key}" محجوز أمنيًا.`);

}

try {

const ctx = AI\_MEMORY.loadSessionContext();

ctx[key] = {

value: typeof value === 'object' ? JSON.stringify(value) : value,

timestamp: Date.now()

};

AI\_MEMORY.saveSessionContext(ctx);

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'setContext:success', { key }); // ✅ تحسين: سجل النجاح

return showMessage("success", `✅ تم حفظ السياق: [${key}]`);

} catch (e) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'setContext:error', { key, error: e.message }); // ✅ تحسين: سجل الخطأ

return showError(`فشل حفظ السياق: ${e.message}`);

}

}

/\*\*

\* @function getContext

\* @description

\* يسترجع قيمة من سياق الجلسة.

\* @param {string} key - المفتاح المراد استرجاع قيمته.

\* @returns {object} - كائن يحتوي على القيمة المسترجعة أو رسالة تحذير.

\*/

function getContext(key) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'getContext:start', { key }); // ✅ تحسين: سجل بداية العملية

const ctx = AI\_MEMORY.loadSessionContext();

const entry = ctx[key];

if (!entry) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'getContext:not\_found', { key }); // ✅ تحسين: سجل عدم العثور

return showMessage("warning", `لا توجد قيمة محفوظة تحت المفتاح "${key}".`);

}

if (entry.expiresAt && Date.now() > entry.expiresAt) {

delete ctx[key];

AI\_MEMORY.saveSessionContext(ctx);

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'getContext:expired', { key }); // ✅ تحسين: سجل انتهاء الصلاحية

return showMessage("warning", `⏱️ انتهت صلاحية المفتاح "${key}" وتم حذفه.`);

}

try {

const parsed = JSON.parse(entry.value);

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'getContext:success\_parsed', { key }); // ✅ تحسين: سجل النجاح (تحليل JSON)

return { type: "context\_value", key, content: parsed };

} catch (e) { // ✅ تم إضافة e للمتغير catch

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'getContext:success\_raw', { key, error: e.message }); // ✅ تحسين: سجل النجاح (قيمة خام) مع الخطأ

return { type: "context\_value", key, content: entry.value };

}

}

/\*\*

\* @function rememberValue

\* @description

\* اسم مستعار لـ setContext.

\* @param {string} key - المفتاح.

\* @param {\*} value - القيمة.

\* @returns {object} - كائن رسالة.

\*/

function rememberValue(key, value) { return setContext(key, value); }

/\*\*

\* @function recallValue

\* @description

\* اسم مستعار لـ getContext.

\* @param {string} key - المفتاح.

\* @returns {object} - كائن رسالة.

\*/

function recallValue(key) { return getContext(key); }

/\*\*

\* @function rememberFor

\* @description

\* يحفظ قيمة في سياق الجلسة لمدة محددة.

\* @param {string} key - المفتاح.

\* @param {\*} value - القيمة.

\* @param {number} minutes - مدة الصلاحية بالدقائق.

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

\*/

function rememberFor(key, value, minutes) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'rememberFor:start', { key, minutes }); // ✅ تحسين: سجل بداية العملية

const ctx = AI\_MEMORY.loadSessionContext();

ctx[key] = {

value: typeof value === 'object' ? JSON.stringify(value) : value,

timestamp: Date.now(),

expiresAt: Date.now() + minutes \* 60 \* 1000

};

AI\_MEMORY.saveSessionContext(ctx);

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'rememberFor:success', { key, minutes }); // ✅ تحسين: سجل النجاح

return showMessage("success", `⏳ تم حفظ [${key}] لمدة ${minutes} دقيقة.`);

}

/\*\*

\* @function clearSessionContext

\* @description

\* يمسح ذاكرة الجلسة بالكامل.

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

\*/

function clearSessionContext() {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'clearSessionContext:start', {}); // ✅ تحسين: سجل بداية العملية

AI\_MEMORY.saveSessionContext({});

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'clearSessionContext:success', {}); // ✅ تحسين: سجل النجاح

return showMessage("success", "🧠 تم مسح ذاكرة الجلسة بالكامل.");

}

/\*\*

\* @function summarizeSession

\* @description

\* يلخص محتويات ذاكرة الجلسة.

\* @returns {object} - كائن يحتوي على ملخص الجلسة.

\*/

function summarizeSession() {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'summarizeSession:start', {}); // ✅ تحسين: سجل بداية العملية

const ctx = AI\_MEMORY.loadSessionContext();

const keys = Object.keys(ctx);

if (keys.length === 0) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'summarizeSession:empty', {}); // ✅ تحسين: سجل الجلسة الفارغة

return showMessage("info", "🧠 لا توجد بيانات في ذاكرة الجلسة.");

}

const lines = keys.map(k => {

const entry = ctx[k];

const expiry = entry.expiresAt ? ` (تنتهي في: ${new Date(entry.expiresAt).toLocaleTimeString()})` : '';

// ✅ تحسين: التأكد من أن القيمة نصية قبل استخدام substring

const valuePreview = String(entry.value).substring(0, 100);

return `- \*\*${k}\*\*${expiry}: \`${valuePreview}\``;

});

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'summarizeSession:success', { keysCount: keys.length }); // ✅ تحسين: سجل النجاح

return { type: "session\_summary", content: `\*\*ملخص ذاكرة الجلسة:\*\*\n${lines.join("\n")}` };

}

// ========== 3. اقتراحات واسترجاع ==========

/\*\*

\* @function suggestNextActions

\* @description

\* يقترح على المستخدم مجموعة من الإجراءات التالية المحتملة.

\* @param {string} jsonSuggestions - سلسلة JSON تحتوي على مصفوفة من الاقتراحات.

\* @returns {object} - كائن يحتوي على الاقتراحات أو رسالة خطأ.

\*/

function suggestNextActions(jsonSuggestions) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'suggestNextActions:start', { json: jsonSuggestions }); // ✅ تحسين: سجل بداية العملية

try {

const arr = JSON.parse(jsonSuggestions);

if (!Array.isArray(arr)) throw new Error("الاقتراحات يجب أن تكون مصفوفة.");

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'suggestNextActions:success', { count: arr.length }); // ✅ تحسين: سجل النجاح

return {

type: 'suggestions',

content: "🧭 ما الخطوة التالية؟",

suggestions: arr

};

} catch (e) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'suggestNextActions:error', { error: e.message }); // ✅ تحسين: سجل الخطأ

return showError(`فشل تحليل الاقتراحات: ${e.message}`);

}

}

/\*\*

\* @function suggestModelSwitch

\* @description

\* يقترح على المستخدم التبديل إلى نموذج Gemini أكثر ملاءمة.

\* @param {string} modelName - اسم النموذج المقترح.

\* @param {string} message - الرسالة التي تظهر للمستخدم مع الاقتراح.

\* @returns {object} - كائن يمثل اقتراح تبديل النموذج.

\*/

function suggestModelSwitch(modelName, message) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'suggestModelSwitch', { model: modelName });

return { type: 'model\_switch\_suggestion', model: modelName, message };

}

/\*\*

\* @function repeatLastPrompt

\* @description

\* يعيد تنفيذ آخر أمر تم إرساله إلى المساعد.

\* @returns {object} - كائن الرد النهائي من AI\_CORE.ask.

\*/

function repeatLastPrompt() {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'repeatLastPrompt:start', {}); // ✅ تحسين: سجل بداية العملية

const ctx = AI\_MEMORY.loadSessionContext();

const prompt = ctx.lastPrompt;

const model = ctx.lastModel || SETTINGS.DEFAULT\_MODEL; // التأكد من أن SETTINGS.DEFAULT\_MODEL موجودة في ENV.SETTINGS

if (!prompt) {

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'repeatLastPrompt:no\_prompt', {}); // ✅ تحسين: سجل عدم وجود أمر سابق

return showMessage("warning", "⏪ لا يوجد أمر سابق محفوظ.");

}

// استدعاء AI\_CORE.ask لإعادة تشغيل الـ pipeline

HELPERS.logOperation('AI\_TOOLS\_DIALOGUE', 'repeatLastPrompt:calling\_ai\_core', { prompt, model }); // ✅ تحسين: سجل استدعاء AI\_CORE

return AI\_CORE.ask(prompt, model);

}

// ========== 📦 التصدير ==========

return {

showMessage,

showError,

askUserConfirmation,

showSummaryTable,

setContext,

getContext,

clearSessionContext,

rememberValue,

recallValue,

rememberFor,

summarizeSession,

suggestNextActions,

suggestModelSwitch,

repeatLastPrompt

};

})(ENV, AI\_MEMORY, HELPERS, AI\_CORE);