/\*\*

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

\* @file ai\_memory.gs - v15.7 (محرك الذاكرة والسجل)

\* @module AI\_MEMORY

\* @description

\* إدارة "ذاكرة" المساعد بنوعيها:

\* - قصيرة المدى (Session Context): تُحفظ في DocumentProperties لتكون دائمة عبر الجلسات.

\* - طويلة المدى (Operations Log): تُحفظ في ورقة "سجل العمليات".

\* تعتمد على: ENV, HELPERS, AI\_CORE, AI\_TOOLS\_DIALOGUE

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

\*/

var AI\_MEMORY = (function(ENV, HELPERS, AI\_CORE, AI\_TOOLS\_DIALOGUE ){ // ✅ حقن التبعيات الصحيحة: ENV, HELPERS, AI\_CORE, AI\_TOOLS\_DIALOGUE

'use strict';

const PROP\_SERVICE = PropertiesService.getDocumentProperties();

const CONTEXT\_KEY = 'AI\_SESSION\_CONTEXT';

// ✅ الوصول إلى SETTINGS مباشرة من ENV

const SETTINGS = ENV.getSetting('SETTINGS'); // تم تغييرها لاستخدام getSetting

// ========== 1. إدارة الذاكرة قصيرة المدى (الجلسة) ==========

/\*\*

\* @function loadSessionContext

\* @description تحمل سياق الجلسة الكامل (بما في ذلك history و lastPrompt) من DocumentProperties.

\* @returns {object} - كائن السياق، أو كائن فارغ مع history فارغة إذا فشل التحميل أو لم يكن موجودًا.

\*/

function loadSessionContext() {

try {

const raw = PROP\_SERVICE.getProperty(CONTEXT\_KEY);

const ctx = raw ? JSON.parse(raw) : {};

// التأكد من وجود history و lastPrompt

ctx.history = Array.isArray(ctx.history) ? ctx.history : [];

ctx.lastPrompt = ctx.lastPrompt || null;

ctx.lastModel = ctx.lastModel || null;

ctx.tempData = ctx.tempData || {}; // لإدارة البيانات المؤقتة

HELPERS.logOperation('AI\_MEMORY', 'loadSessionContext:success', { keys: Object.keys(ctx) });

return ctx;

} catch (e) {

HELPERS.logOperation('AI\_MEMORY', 'loadSessionContext:error', { error: e.message });

return { history: [], lastPrompt: null, lastModel: null, tempData: {} };

}

}

/\*\*

\* @function saveSessionContext

\* @description تحفظ سياق الجلسة الكامل في DocumentProperties.

\* @param {object} context - كائن السياق المراد حفظه.

\*/

function saveSessionContext(context) {

try {

PROP\_SERVICE.setProperty(CONTEXT\_KEY, JSON.stringify(context));

HELPERS.logOperation('AI\_MEMORY', 'saveSessionContext:success', { keys: Object.keys(context) });

} catch (e) {

HELPERS.logOperation('AI\_MEMORY', 'saveSessionContext:error', { error: e.message });

AI\_TOOLS\_DIALOGUE.showError(`فشل حفظ سياق الجلسة: ${e.message}`); // ✅ استخدام AI\_TOOLS\_DIALOGUE

}

}

/\*\*

\* @function clearSessionContext

\* @description تمسح سياق الجلسة بالكامل من DocumentProperties.

\*/

function clearSessionContext() {

try {

PROP\_SERVICE.deleteProperty(CONTEXT\_KEY);

HELPERS.logOperation('AI\_MEMORY', 'clearSessionContext:success', {});

AI\_TOOLS\_DIALOGUE.showMessage("info", "✅ تم مسح سياق الجلسة بنجاح."); // ✅ استخدام AI\_TOOLS\_DIALOGUE

} catch (e) {

HELPERS.logOperation('AI\_MEMORY', 'clearSessionContext:error', { error: e.message });

AI\_TOOLS\_DIALOGUE.showError(`فشل مسح سياق الجلسة: ${e.message}`); // ✅ استخدام AI\_TOOLS\_DIALOGUE

}

}

/\*\*

\* @function addMessageToHistory

\* @description تضيف رسالة إلى سجل المحادثة في سياق الجلسة.

\* @param {string} role - دور المرسل ('user' أو 'model' أو 'tool').

\* @param {string|object} content - محتوى الرسالة (نص أو كائن).

\*/

function addMessageToHistory(role, content) {

const ctx = loadSessionContext();

const messageContent = typeof content === 'object' ? JSON.stringify(content) : content;

ctx.history.push({ role: role, parts: [{ text: messageContent }] });

// الحفاظ على حجم السجل ضمن حد معين (مثلاً آخر 20 رسالة)

const MAX\_HISTORY\_LENGTH = ENV.getSetting('MAX\_HISTORY\_LENGTH') || 20; // ✅ استخدام ENV.getSetting

if (ctx.history.length > MAX\_HISTORY\_LENGTH) {

ctx.history = ctx.history.slice(-MAX\_HISTORY\_LENGTH);

}

saveSessionContext(ctx);

HELPERS.logOperation('AI\_MEMORY', 'addMessageToHistory:success', { role, contentPreview: messageContent.substring(0, Math.min(messageContent.length, 50)) });

}

/\*\*

\* @function getSessionHistory

\* @description تُرجع سجل المحادثة الحالي.

\* @returns {Array<object>} - مصفوفة من كائنات الرسائل.

\*/

function getSessionHistory() {

return loadSessionContext().history;

}

/\*\*

\* @function setLastPrompt

\* @description تحفظ آخر أمر تم إرساله إلى النموذج.

\* @param {string} prompt - نص الأمر.

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

\*/

function setLastPrompt(prompt, model) {

const ctx = loadSessionContext();

ctx.lastPrompt = prompt;

ctx.lastModel = model;

saveSessionContext(ctx);

HELPERS.logOperation('AI\_MEMORY', 'setLastPrompt:success', { promptPreview: prompt.substring(0, Math.min(prompt.length, 50)), model });

}

/\*\*

\* @function setTempData

\* @description تحفظ بيانات مؤقتة في سياق الجلسة مع تاريخ انتهاء صلاحية.

\* @param {string} key - المفتاح لربط البيانات.

\* @param {any} value - القيمة المراد حفظها.

\* @param {number} [ttlSeconds=300] - مدة صلاحية البيانات بالثواني (افتراضي: 5 دقائق).

\*/

function setTempData(key, value, ttlSeconds = 300) {

const ctx = loadSessionContext();

ctx.tempData[key] = {

value: value,

expiresAt: Date.now() + (ttlSeconds \* 1000)

};

saveSessionContext(ctx);

HELPERS.logOperation('AI\_MEMORY', 'setTempData:success', { key, ttlSeconds });

}

/\*\*

\* @function getTempData

\* @description تسترجع البيانات المؤقتة من سياق الجلسة، وتزيلها إذا انتهت صلاحيتها.

\* @param {string} key - المفتاح لاسترجاع البيانات.

\* @returns {any|null} - القيمة المحفوظة أو null إذا لم تكن موجودة أو انتهت صلاحيتها.

\*/

function getTempData(key) {

const ctx = loadSessionContext();

const entry = ctx.tempData[key];

if (entry) {

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

delete ctx.tempData[key]; // إزالة البيانات منتهية الصلاحية

saveSessionContext(ctx);

HELPERS.logOperation('AI\_MEMORY', 'getTempData:expired', { key });

return null;

}

HELPERS.logOperation('AI\_MEMORY', 'getTempData:success', { key });

return entry.value;

}

HELPERS.logOperation('AI\_MEMORY', 'getTempData:not\_found', { key });

return null;

}

/\*\*

\* @function getRecentOperationsSummary

\* @description

\* تُرجع ملخصًا لآخر N عملية مسجلة في ورقة "سجل العمليات".

\* @param {number} count - عدد العمليات المراد تلخيصها.

\* @returns {string} - ملخص نصي للعمليات.

\*/

function getRecentOperationsSummary(count) {

try {

const sheetName = ENV.getSetting('OPERATION\_LOG\_SHEET\_NAME');

const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(sheetName);

if (!sheet || sheet.getLastRow() < 2) {

return "لا توجد عمليات مسجلة حديثًا.";

}

const lastRow = sheet.getLastRow();

const startRow = Math.max(2, lastRow - count + 1); // ابدأ من الصف الثاني (بعد الرؤوس)

const data = sheet.getRange(startRow, 1, lastRow - startRow + 1, sheet.getLastColumn()).getValues();

const summaryLines = data.map(row => {

const timestamp = Utilities.formatDate(row[0], Session.getScriptTimeZone(), ENV.getDateFormat() + " HH:mm");

const module = row[1];

const action = row[2];

const details = row[3]; // الملخص/التفاصيل

return `- [${timestamp}] ${module} - ${action}: ${String(details).substring(0, 100)}...`;

});

return summaryLines.join("\\n");

} catch (e) {

HELPERS.logOperation('AI\_MEMORY', 'getRecentOperationsSummary:error', { error: e.message });

return "فشل جلب ملخص العمليات.";

}

}

// ========== 2. إدارة الذاكرة طويلة المدى (سجل العمليات) ==========

/\*\*

\* @function logOperation

\* @description

\* 🧠 يسجل العمليات في سجل الأداء طويل المدى (ورقة OperationLog).

\* هذه هي الدالة الموحدة للتسجيل.

\* @param {string} moduleName - اسم الوحدة التي تقوم بالتسجيل.

\* @param {string} action - وصف مختصر للفعل المنفذ.

\* @param {any} summary - ملخص التنفيذ أو الكائن المرتبط.

\*/

function logOperation(moduleName, action, summary) {

try {

const sheetName = ENV.getSetting('OPERATION\_LOG\_SHEET\_NAME');

const sheet = HELPERS.getSheet(sheetName, ["Timestamp", "Module", "Action", "Summary"]); // التأكد من وجود الورقة والرؤوس

const timestamp = new Date();

const summaryText = typeof summary === 'object' ? JSON.stringify(summary) : String(summary);

sheet.appendRow([timestamp, moduleName, action, summaryText]);

} catch (e) {

console.error(`AI\_MEMORY.logOperation failed: ${e.message}`);

}

}

/\*\*

\* @function repeatLastPrompt

\* @description تعيد تنفيذ آخر طلب تم حفظه في الجلسة عن طريق استدعاء AI\_CORE.ask().

\* @returns {object} – نتيجة استدعاء AI\_CORE.ask() أو رسالة معلوماتية من AI\_TOOLS\_DIALOGUE إذا لم يكن هناك أمر سابق.

\*/

function repeatLastPrompt() {

HELPERS.logOperation('AI\_MEMORY', 'repeatLastPrompt:start ', {}); // ✅ تسجيل بداية العملية

const ctx = loadSessionContext();

if (!ctx.lastPrompt) {

HELPERS.logOperation('AI\_MEMORY', 'repeatLastPrompt:no\_prompt ', {}); // ✅ تسجيل عدم وجود أمر سابق

return AI\_TOOLS\_DIALOGUE.showMessage("warning", "⚠️ لا يوجد أمر سابق محفوظ في هذه الجلسة."); // ✅ استخدام AI\_TOOLS\_DIALOGUE

}

HELPERS.logOperation('AI\_MEMORY', 'repeatLastPrompt:calling\_ai\_core', { prompt: ctx.lastPrompt, model: ctx.lastModel }); // ✅ تسجيل الاستدعاء

return AI\_CORE.ask(ctx.lastPrompt, ctx.lastModel ); // ✅ استخدام AI\_CORE

}

// --- تصدير الواجهة العامة للوحدة ---

return {

loadSessionContext,

saveSessionContext,

clearSessionContext,

addMessageToHistory,

getSessionHistory,

setLastPrompt,

setTempData,

getTempData,

getRecentOperationsSummary, // تصدير الدالة الجديدة

logOperation, // تصدير دالة التسجيل الموحدة

repeatLastPrompt

};

})(

GAssistant.Config, // تم تغيير ENV إلى GAssistant.Config

GAssistant.Helpers,

GAssistant.AI.Core, // افتراض وجود هذه الوحدة

GAssistant.AI.Dialogue // افتراض وجود هذه الوحدة

);