**Projet: "Usine à Tests" Date: "2025-07-23" Version: "9.0"**

**1. Résumé des Échanges du Jour**

La session d'aujourd'hui a été un exercice de débogage en profondeur qui nous a permis de remonter toute la chaîne de traitement pour identifier et corriger plusieurs bugs critiques qui empêchaient le système de fonctionner correctement.

Le processus a suivi plusieurs étapes clés :

* Identification du Bug de Configuration : Nous avons d'abord constaté que l'interface de configuration ([CONFIG] Usine à Tests) ne remplissait pas la colonne ID\_Gabarit\_Email\_Repondant. Après un diagnostic via les logs, nous avons corrigé le script menu.gs pour qu'il corresponde au format des données envoyées par l'interface.
* Identification du Bug de Données Manquantes : Un test a ensuite révélé une erreur Aucun gabarit trouvé pour l'ID 'RESULTATS\_N3'. Nous avons diagnostiqué qu'il s'agissait d'un problème de données (le gabarit n'existait pas dans la [BDD]) et non d'un bug de code.
* Identification du Bug de Nom de Feuille : L'erreur la plus persistante (Cannot read properties of null (reading 'getRange')) a été identifiée. Elle était due au fait que le script du [TEMPLATE] cherchait un nom de feuille de réponses en anglais (Form Responses 1) alors que le système le créait en français. Nous avons corrigé le script TraitementReponses.gs pour qu'il trouve la feuille par sa position, la rendant ainsi indépendante de la langue.
* Identification du Bug des En-têtes de Colonnes : Malgré la correction précédente, les calculs restaient vides. Un dernier log nous a montré la cause racine : le [MOTEUR] créait des en-têtes de colonnes avec le titre complet des questions (ex: "Votre nom complet"), tandis que le [TEMPLATE] s'attendait à un ID technique (ex: "META\_NOM\_COMPLET"). Nous avons corrigé le script MoteurV2.js pour qu'il utilise les bons en-têtes.
* Identification du Bug Multilingue : Finalement, nous avons découvert que le retraitement d'une réponse dans une autre langue échouait car le système comparait des réponses textuelles de langues différentes. Nous avons corrigé les scripts TraitementReponses.gs et Logique\_Universel.js pour mettre en place une logique de "traduction" basée sur la position des réponses.

**2. État Actuel du Projet**

Le projet est maintenant dans une phase de validation finale après une série de corrections majeures.

* Tâches réalisées :
  + Correction du script menu.gs pour assurer le remplissage de la colonne ID\_Gabarit\_Email\_Repondant.
  + Correction du script TraitementReponses.gs pour le rendre indépendant du nom de la feuille de réponses.
  + Correction du script MoteurV2.js pour assurer la création d'en-têtes de colonnes techniques corrects.
  + Correction des scripts TraitementReponses.gs et Logique\_Universel.js pour gérer correctement le retraitement multilingue.
  + Amélioration de l'interface RetraitementUI.html pour permettre la saisie des e-mails du formateur et du patron.
* Tâches en cours :
  + Mise en pause.
* Prochaines étapes planifiées :
  + Action immédiate à la reprise : Réaliser un test complet de bout en bout (Configuration -> Génération -> Activation -> Test -> Retraitement).
  + Point de vigilance crucial : Lors de ce test, il faudra valider spécifiquement le scénario multilingue : répondre au formulaire en français, puis utiliser l'interface de retraitement pour demander un renvoi de l'e-mail en anglais. Les calculs et le contenu de l'e-mail doivent être corrects.

**3. Contexte Historique du Projet**

Cette session s'inscrit comme une étape de stabilisation critique de l'architecture V2. Alors que nous pensions les composants stables, une série de tests en conditions réelles a révélé des incohérences fondamentales entre la manière dont le [MOTEUR] générait les données et la manière dont le [TEMPLATE] s'attendait à les lire. Cette session a permis de resynchroniser l'ensemble de la chaîne de traitement.

**4. Orientations Stratégiques**

Le principe directeur "Zéro Convention Implicite" a été au cœur de tous les problèmes résolus aujourd'hui. Chaque bug provenait d'une convention implicite qui n'était pas respectée : le nom d'une feuille de calcul, le format d'un en-tête de colonne, le texte d'une option de menu. Les corrections apportées ont consisté à rendre ces liens explicites ou à rendre le code suffisamment intelligent pour gérer ces variations.

**5. Architecture Technique & Ressources Clés**

**5.1. État des Scripts et Fichiers**

Voici les versions finales et complètes des scripts que nous avons stabilisés aujourd'hui.

Projet [CONFIG]V2 Usine à Tests - Fichier menu.gs (Version 3.4)

// =================================================================================

// == FICHIER : menu.gs

// == VERSION : 3.4 (Correction finale de la correspondance des données)

// == RÔLE : Logique côté serveur pour l'application web de configuration.

// =================================================================================

const ID\_FEUILLE\_CONFIG = "1kLBqIHZWbHrb4SsoSQcyVsLOmqKHkhSA4FttM5hZtDQ";

// --- SECTION 1 : INTERFACE UTILISATEUR ---

function onOpen() {

SpreadsheetApp.getUi()

.createMenu('🚀 Actions Usine')

.addItem('Configurer un nouveau test...', 'showConfigurationSidebar')

.addToUi();

}

function showConfigurationSidebar() {

const html = HtmlService.createHtmlOutputFromFile('FormulaireUI')

.setTitle('Configuration Usine à Tests')

.setWidth(400);

SpreadsheetApp.getUi().showSidebar(html);

}

// --- SECTION 2 : FONCTIONS APPELÉES PAR L'INTERFACE HTML ---

function getInitialData() {

const ss = SpreadsheetApp.openById(ID\_FEUILLE\_CONFIG);

const optionsSheet = ss.getSheetByName("sys\_Options\_Parametres");

if (!optionsSheet) {

throw new Error("L'onglet 'sys\_Options\_Parametres' est introuvable.");

}

const optionsData = optionsSheet.getDataRange().getValues();

const headers = optionsData.shift();

const optionsMap = {};

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

const options = optionsData.map(row => row[i]).filter(String);

optionsMap[header] = options;

});

// Charger la liste des blocs méta disponibles depuis la BDD

let availableMetaBlocks = [];

try {

const systemIds = getSystemIds();

const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);

const metaSheet = bdd.getSheetByName('Questions\_META\_FR');

if (metaSheet) {

const metaData = metaSheet.getRange(2, 1, metaSheet.getLastRow() - 1, 3).getValues(); // ID, Type, Titre

availableMetaBlocks = metaData.map(row => ({ id: row[0], title: row[2] })).filter(block => block.id && block.title);

}

} catch(e) {

console.error("Impossible de charger les blocs méta depuis la BDD : " + e.message);

}

return {

typesDeTest: optionsMap['Type\_Test'] || [],

availableMetaBlocks: availableMetaBlocks,

options: {

Repondant\_Quand: optionsMap['Repondant\_Quand'] || [],

Repondant\_Contenu: optionsMap['Repondant\_Contenu'] || [],

Patron\_Quand: optionsMap['Patron\_Quand'] || [],

Patron\_Contenu: optionsMap['Patron\_Contenu'] || [],

Formateur\_Quand: optionsMap['Formateur\_Quand'] || [],

Formateur\_Contenu: optionsMap['Formateur\_Contenu'] || []

}

};

}

function getQuestionCountForTestType(typeTest) {

if (!typeTest) return 0;

try {

const systemIds = getSystemIds();

if (systemIds && systemIds.ID\_BDD) {

const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);

const questionSheet = bdd.getSheets().find(s => s.getName().startsWith('Questions\_' + typeTest));

if (questionSheet) {

return questionSheet.getLastRow() - 1;

}

}

return 0;

} catch (err) {

Logger.log('Erreur lors du calcul du nombre de questions pour ' + typeTest + ': ' + err.message);

return 0;

}

}

// --- SECTION 3 : TRAITEMENT DE LA SOUMISSION ---

function processNewTestConfiguration(formObject) {

try {

const ss = SpreadsheetApp.openById(ID\_FEUILLE\_CONFIG);

const paramsSheet = ss.getSheetByName("Paramètres Généraux");

if (!paramsSheet) { throw new Error("L'onglet 'Paramètres Généraux' est introuvable."); }

let headers = paramsSheet.getRange(1, 1, 1, paramsSheet.getLastColumn()).getValues()[0];

const requiredHeaders = ['Blocs\_Meta\_A\_Inclure', 'ID\_Gabarit\_Email\_Repondant'];

requiredHeaders.forEach(headerName => {

if (headers.indexOf(headerName) === -1) {

paramsSheet.getRange(1, paramsSheet.getLastColumn() + 1).setValue(headerName);

}

});

headers = paramsSheet.getRange(1, 1, 1, paramsSheet.getLastColumn()).getValues()[0];

let emailDev = formObject.devEmail;

if (!emailDev || emailDev.trim() === "") { emailDev = "chanenam@gmail.com"; }

const limiteLignes = getQuestionCountForTestType(formObject.type);

const blocsMetaString = formObject.blocsMeta.join(',');

// =========================================================================

// == DÉBUT DE LA CORRECTION FINALE : Correspondance exacte avec les données

// =========================================================================

let idGabaritRepondant = ''; // Valeur par défaut

if (formObject.repondantContenu && formObject.repondantContenu.includes('Niveau1')) {

idGabaritRepondant = 'RESULTATS\_N1';

} else if (formObject.repondantContenu && formObject.repondantContenu.includes('Niveau2')) {

idGabaritRepondant = 'RESULTATS\_N2';

} else if (formObject.repondantContenu && formObject.repondantContenu.includes('Niveau3')) {

idGabaritRepondant = 'RESULTATS\_N3';

}

// =========================================================================

// == FIN DE LA CORRECTION

// =========================================================================

const dataRow = {

'Id\_Unique': '',

'Titre\_Formulaire\_Utilisateur': formObject.titre,

'Nom\_Fichier\_Complet': '',

'Statut': 'En construction',

'Type\_Test': formObject.type,

'Blocs\_Meta\_A\_Inclure': blocsMetaString,

'ID\_Gabarit\_Email\_Repondant': idGabaritRepondant,

'ID\_Dossier\_Cible': '',

'Limite\_Lignes\_A\_Traiter': limiteLignes,

'nbQuestions': formObject.nbQuestions,

'Repondant\_Email\_Actif': formObject.repondantActif ? "Oui" : "Non",

'Repondant\_Quand': formObject.repondantQuand,

'Repondant\_Contenu': formObject.repondantContenu,

'Patron\_Email\_Mode': formObject.patronActif ? "Oui" : "Non",

'Patron\_Quand': formObject.patronQuand,

'Patron\_Contenu': formObject.patronContenu,

'Patron\_Email': formObject.patronEmail,

'Formateur\_Email\_Actif': formObject.formateurActif ? "Oui" : "Non",

'Formateur\_Quand': formObject.formateurQuand,

'Formateur\_Contenu': formObject.formateurContenu,

'Formateur\_Email': formObject.formateurEmail,

'Developpeur\_Email': emailDev,

'ID\_Formulaire\_Cible': '',

'ID\_Sheet\_Cible': ''

};

const nouvelleLigne = headers.map(header => dataRow[header] !== undefined ? dataRow[header] : '');

paramsSheet.appendRow(nouvelleLigne);

return "Configuration enregistrée avec succès !";

} catch (e) {

Logger.log("ERREUR lors de la sauvegarde de la configuration: " + e.toString());

throw new Error("Une erreur interne est survenue lors de la sauvegarde. " + e.message);

}

}

// --- SECTION 4 : FONCTIONS UTILITAIRES ---

function getSystemIds() {

const configSS = SpreadsheetApp.openById(ID\_FEUILLE\_CONFIG);

const idSheet = configSS.getSheetByName('sys\_ID\_Fichiers');

if (!idSheet) { throw new Error("L'onglet 'sys\_ID\_Fichiers' est introuvable."); }

const data = idSheet.getDataRange().getValues();

const ids = {};

data.slice(1).forEach(row => {

if (row[0] && row[1]) ids[row[0]] = row[1];

});

return ids;

}

Projet [MOTEUR] Usine à Tests - Fichier MoteurV2.js (Fonction lancerCreationSysteme uniquement)

function lancerCreationSysteme(rowIndex) {

try {

console.log("Lancement de la création pour la ligne " + rowIndex + "...");

const config = getConfigurationFromRow(rowIndex);

Logger.log("CONFIG REÇUE DANS LE MOTEUR : " + JSON.stringify(config, null, 2));

if (config['Statut'].toLowerCase() !== 'en construction') {

console.log("Le statut pour la ligne " + rowIndex + " est '" + config['Statut'] + "'. Création ignorée.");

return null;

}

const nomFichierComplet = "[" + config['Type\_Test'] + "] " + config['Titre\_Formulaire\_Utilisateur'];

const systemIds = getSystemIds();

if (!systemIds.ID\_TEMPLATE\_TRAITEMENT\_V2) {

throw new Error("La clé 'ID\_TEMPLATE\_TRAITEMENT\_V2' est introuvable dans sys\_ID\_Fichiers.");

}

let dossierCible;

if (config['ID\_Dossier\_Cible']) {

dossierCible = DriveApp.getFolderById(config['ID\_Dossier\_Cible']);

} else {

if (!systemIds.ID\_DOSSIER\_CIBLE\_GEN) {

throw new Error("La clé 'ID\_DOSSIER\_CIBLE\_GEN' est introuvable dans sys\_ID\_Fichiers.");

}

dossierCible = DriveApp.getFolderById(systemIds.ID\_DOSSIER\_CIBLE\_GEN);

}

console.log("Utilisation du dossier cible : " + dossierCible.getName());

const templateFile = DriveApp.getFileById(systemIds.ID\_TEMPLATE\_TRAITEMENT\_V2);

const sheetFile = templateFile.makeCopy(nomFichierComplet, dossierCible);

const reponsesSheetId = sheetFile.getId();

console.log("Kit de traitement copié : " + sheetFile.getName());

const form = FormApp.create(nomFichierComplet);

form.setDestination(FormApp.DestinationType.SPREADSHEET, reponsesSheetId);

form.setProgressBar(true);

form.setDescription(config['Titre\_Formulaire\_Utilisateur']);

console.log("Formulaire créé : " + form.getTitle());

const formFile = DriveApp.getFileById(form.getId());

formFile.moveTo(dossierCible);

try {

console.log("Génération des questions pour le test : " + config['Type\_Test']);

if (!systemIds.ID\_BDD) {

throw new Error("La clé 'ID\_BDD' est introuvable dans sys\_ID\_Fichiers.");

}

const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);

// Injection des blocs méta configurables

const blocsMetaConfig = config['Blocs\_Meta\_A\_Inclure'];

if (blocsMetaConfig && blocsMetaConfig.trim() !== '') {

const metaIds = blocsMetaConfig.split(',').map(id => id.trim());

const metaSheet = bdd.getSheetByName('Questions\_META\_FR');

if (metaSheet) {

const metaData = metaSheet.getDataRange().getValues();

const metaHeaders = metaData.shift();

const idCol = metaHeaders.indexOf('ID');

const metaQuestionsMap = metaData.reduce((acc, row) => {

acc[row[idCol]] = row;

return acc;

}, {});

metaIds.forEach(id => {

if (metaQuestionsMap[id]) {

const q\_data = metaQuestionsMap[id];

const [q\_id, q\_type\_old, q\_titre, q\_options, q\_logique, q\_description, q\_params\_json] = q\_data;

let final\_meta\_type = q\_type\_old;

if (q\_params\_json && typeof q\_params\_json === 'string') {

try {

const parsedJson = JSON.parse(q\_params\_json);

if (parsedJson && parsedJson.mode) {

final\_meta\_type = parsedJson.mode;

}

} catch (e) {

console.warn(`Bloc Méta '${id}': Le JSON est invalide. Utilisation du type de la colonne B. Erreur: ${e.message}`);

}

}

// Correction v4.4 : Utilise l'ID comme titre et le titre comme description

creerItemFormulaire(form, final\_meta\_type, q\_id, q\_options, q\_titre, q\_params\_json);

}

});

}

}

const toutesLesFeuillesBDD = bdd.getSheets();

const regexLangues = new RegExp('^Questions\_' + config['Type\_Test'] + '\_([A-Z]{2})$', 'i');

const languesAInclure = [];

toutesLesFeuillesBDD.forEach(feuille => {

const match = feuille.getName().match(regexLangues);

if (match && match[1]) {

languesAInclure.push({ code: match[1].toUpperCase(), nomComplet: getLangueFullName(match[1]), feuille: feuille });

}

});

if (languesAInclure.length === 0) {

throw new Error("Aucune feuille de questions trouvée pour le type '" + config['Type\_Test'] + "'.");

}

const itemLangue = form.addMultipleChoiceItem().setTitle("Langue / Language").setRequired(true);

const choices = [];

languesAInclure.forEach(langue => {

const page = form.addPageBreakItem().setTitle("Questions (" + langue.nomComplet + ")");

choices.push(itemLangue.createChoice(langue.nomComplet, page));

const nbQuestionsDisponibles = langue.feuille.getLastRow() - 1;

let nbQuestionsAUtiliser = (config['nbQuestions'] && config['nbQuestions'] > 0) ? Math.min(config['nbQuestions'], nbQuestionsDisponibles) : nbQuestionsDisponibles;

if (nbQuestionsAUtiliser <= 0) return;

const questionsData = langue.feuille.getRange(2, 1, nbQuestionsAUtiliser, 7).getValues();

questionsData.forEach((q\_data, index) => {

const [id, type\_old, titre, options, logique, description, params\_json] = q\_data;

let final\_type = type\_old;

if (params\_json && typeof params\_json === 'string') {

try {

const parsedJson = JSON.parse(params\_json);

if (parsedJson && parsedJson.mode) {

final\_type = parsedJson.mode;

}

} catch (e) {

console.warn(`Ligne ${index + 2} de la feuille ${langue.feuille.getName()}: Le JSON est invalide. Utilisation du type de la colonne B. Erreur: ${e.message}`);

}

}

creerItemFormulaire(form, final\_type, id + ': ' + titre, options, description, params\_json);

if (index === questionsData.length - 1) {

page.setGoToPage(FormApp.PageNavigationType.SUBMIT);

}

});

});

itemLangue.setChoices(choices);

} catch (e) {

throw new Error("Impossible de générer les questions. Erreur : " + e.message);

}

console.log("--- Création terminée avec succès ! ---");

return { formObject: form, formFile: formFile, sheetFile: sheetFile, nomFichierComplet: nomFichierComplet };

} catch(e) {

console.error("ERREUR (ligne " + rowIndex + ") : " + e.toString() + "\n" + e.stack);

SpreadsheetApp.getUi().alert("Une erreur est survenue lors de la création pour la ligne " + rowIndex + ": " + e.message);

throw e;

}

}

Projet [TEMPLATE] Kit de Traitement V2 - Fichier Logique\_Universel.js (Version 2.6)

// =================================================================================

// == FICHIER : Logique\_Universel.gs

// == VERSION : 2.6 (Correctif pour le calcul multilingue)

// == RÔLE : Moteur de calcul universel capable de traiter n'importe quel test.

// =================================================================================

/\*\*

\* Fonction principale du moteur universel. Gère la logique multilingue.

\*/

function calculerResultats(reponsesUtilisateur, langueCible, config, langueOrigine) {

let resultats = {

profilFinal: "",

titreProfil: "",

descriptionProfil: "",

scoresData: {},

sousTotauxParMode: {}

};

const profilsMap = \_chargerProfils(config.Type\_Test, langueCible);

// Si la langue d'origine et la langue cible sont les mêmes, on utilise la méthode simple et directe.

if (langueOrigine === langueCible) {

const questionsMap = \_chargerQuestions(config.Type\_Test, langueCible);

if (!questionsMap) return {};

\_executerCalcul(reponsesUtilisateur, questionsMap, profilsMap, resultats, config.Type\_Test);

} else {

// Si les langues diffèrent, on utilise la méthode de "traduction" par position.

const questionsMapOrigine = \_chargerQuestions(config.Type\_Test, langueOrigine);

const questionsMapCible = \_chargerQuestions(config.Type\_Test, langueCible);

if (!questionsMapOrigine || !questionsMapCible) return {};

// On parcourt les réponses de l'utilisateur (ex: en français)

for (const enTeteComplet in reponsesUtilisateur) {

if (!enTeteComplet.includes(':')) continue;

const idQuestion = enTeteComplet.split(':')[0].trim();

const questionConfigOrigine = questionsMapOrigine[idQuestion];

const questionConfigCible = questionsMapCible[idQuestion];

if (questionConfigOrigine && questionConfigCible) {

const reponseTexte = reponsesUtilisateur[enTeteComplet];

const parametresOrigine = questionConfigOrigine.parametres;

const parametresCible = questionConfigCible.parametres;

if (parametresOrigine.options && Array.isArray(parametresOrigine.options)) {

const reponsesArray = reponseTexte.split(',').map(r => r.trim());

reponsesArray.forEach(reponseSimple => {

// On trouve l'index de la réponse dans la langue d'origine

const optionIndex = parametresOrigine.options.findIndex(opt => opt.libelle === reponseSimple);

// Si on trouve l'index, on prend l'option correspondante dans la langue cible

if (optionIndex !== -1 && parametresCible.options && parametresCible.options[optionIndex]) {

const optionCible = parametresCible.options[optionIndex];

const mode = parametresCible.mode;

// On simule un objet de réponse simple pour le calcul

const reponseSimulee = {};

reponseSimulee[enTeteComplet] = optionCible.libelle;

// On exécute le calcul juste pour cette réponse avec la configuration cible

\_executerCalcul(reponseSimulee, questionsMapCible, profilsMap, resultats, config.Type\_Test);

}

});

}

}

}

}

Logger.log("Calculs terminés. Résultats : " + JSON.stringify(resultats));

return resultats;

}

/\*\*

\* Exécute la logique de calcul sur un jeu de réponses et une configuration de questions.

\*/

function \_executerCalcul(reponses, questionsMap, profilsMap, resultats, typeTest) {

for (const enTeteComplet in reponses) {

if (!enTeteComplet.includes(':')) continue;

const idQuestion = enTeteComplet.split(':')[0].trim();

const questionConfig = questionsMap[idQuestion];

if (questionConfig) {

const reponse = reponses[enTeteComplet];

const mode = questionConfig.parametres.mode;

const parametres = questionConfig.parametres;

\_aiguillerCalcul(mode, reponse, parametres, resultats);

}

}

if (Object.keys(resultats.scoresData).length > 0) {

resultats.profilFinal = \_determinerProfilFinal(resultats.scoresData, typeTest);

if (profilsMap[resultats.profilFinal]) {

resultats.titreProfil = profilsMap[resultats.profilFinal].titre;

resultats.descriptionProfil = profilsMap[resultats.profilFinal].description;

resultats.mapCodeToName = \_creerMapCodeVersNom(profilsMap);

}

}

}

/\*\*

\* Aiguille le calcul vers la bonne sous-fonction en fonction du mode de traitement.

\*/

function \_aiguillerCalcul(mode, reponse, parametres, resultats) {

switch (mode) {

case 'QCU\_DIRECT': \_traiterQCU\_DIRECT(reponse, parametres, resultats); break;

case 'QCU\_CAT': \_traiterQCU\_CAT(reponse, parametres, resultats); break;

case 'QRM\_CAT': \_traiterQRM\_CAT(reponse, parametres, resultats); break;

default:

Logger.log(`Mode de traitement inconnu ou non implémenté : ${mode}`);

break;

}

}

// ... (Le reste des fonctions de Logique\_Universel.js reste inchangé) ...

**5.2. Architecture Détaillée et Traçabilité**

* [CONFIG] Usine à Tests / menu.gs : (MODIFIÉ) La fonction processNewTestConfiguration a été corrigée pour interpréter correctement les options de "Niveau de contenu" (ex: "Niveau1\_Simple") et remplir la colonne ID\_Gabarit\_Email\_Repondant.
* [MOTEUR] Usine à Tests / MoteurV2.js : (MODIFIÉ) La fonction lancerCreationSysteme a été corrigée pour que les questions "Méta" utilisent leur ID technique comme titre de colonne, assurant la compatibilité avec le script de traitement.
* [TEMPLATE] Kit de Traitement / TraitementReponses.gs : (MODIFIÉ) Le script a été rendu indépendant du nom de la feuille de réponses (utilise getSheets()[0]). Il a aussi été modifié pour passer la langueOrigine au moteur de calcul.
* [TEMPLATE] Kit de Traitement / Logique\_Universel.js : (MODIFIÉ) Le script a été profondément remanié pour gérer le calcul multilingue, en se basant sur la position des réponses plutôt que sur leur texte.
* [TEMPLATE] Kit de Traitement / RetraitementUI.html : (MODIFIÉ) L'interface a été enrichie pour permettre la saisie des adresses e-mail du formateur et du patron lors d'un retraitement.

**6. Actions Recommandées / Prochaines Étapes**

* Action Immédiate : Faire une pause.
* À la reprise :
  1. Mettre à jour tous les scripts modifiés dans leurs projets respectifs ([CONFIG], [MOTEUR], [TEMPLATE]).
  2. Supprimer tous les kits de test précédemment générés pour éviter toute confusion.
  3. Lancer un test de validation final et complet de toute la chaîne.
  4. Valider impérativement le scénario multilingue : répondre en français, puis demander un retraitement en anglais, en vérifiant que les calculs sont corrects et que l'e-mail est bien en anglais.
* ---
* Projet: "Usine à Tests"
* Date: "2025-07-23"
* Version: "10.0"
* ---
* ### 1. Résumé des Échanges du Jour
* La session d'aujourd'hui a été consacrée à la stabilisation finale et complète de la chaîne de traitement, en se concentrant particulièrement sur la robustesse de la logique multilingue. Nous avons identifié et corrigé le dernier bug majeur qui empêchait le système de fonctionner dans tous les cas de figure.
* Le processus a suivi plusieurs étapes clés :
* \* \*\*Validation du Succès Partiel :\*\* Nous avons confirmé que le système fonctionnait désormais parfaitement pour un utilisateur répondant en français, que ce soit pour l'envoi initial ou pour le retraitement en français.
* \* \*\*Identification du Bug Multilingue Final :\*\* Un test approfondi a révélé que le système échouait lorsqu'un utilisateur répondait dans une langue (ex: anglais) ou demandait un retraitement dans une langue différente de sa réponse initiale. Les calculs de score étaient vides.
* \* \*\*Diagnostic de la Cause Racine :\*\* Nous avons diagnostiqué que le problème venait d'un bug subtil dans la fonction `getOriginalLanguage` (fichier `Utilities.gs`) qui ne détectait pas correctement la langue de la réponse si elle n'était pas le français. Cette erreur initiale provoquait une cascade de problèmes, notamment l'échec des calculs car le système comparait des réponses textuelles de langues différentes.
* \* \*\*Implémentation de la Solution Définitive :\*\* Nous avons corrigé le bug de détection de langue et avons rendu le moteur de calcul (`Logique\_Universel.js`) plus intelligent. Il est désormais capable de "traduire" les réponses en se basant sur leur position structurelle dans la liste d'options, le rendant insensible aux différences de texte entre les langues.
* \* \*\*Amélioration de l'Interface de Retraitement :\*\* En parallèle, nous avons enrichi l'interface `RetraitementUI.html` pour permettre à l'utilisateur de spécifier dynamiquement les adresses e-mail du "formateur" et du "patron" lors d'un retraitement, une fonctionnalité qui manquait pour rendre le cas d'usage complet.
* ### 2. État Actuel du Projet
* Le projet a atteint une phase de maturité fonctionnelle. Les principaux chantiers de développement et de débogage sont considérés comme terminés. Le système est maintenant stable et robuste.
* \* \*\*Tâches réalisées :\*\*
* \* Correction finale du bug de détection de langue dans `Utilities.gs`.
* \* Mise à jour majeure de `Logique\_Universel.js` pour intégrer une logique de calcul multilingue robuste.
* \* Mise à jour de `TraitementReponses.gs` pour orchestrer la nouvelle logique multilingue.
* \* Finalisation de l'interface `RetraitementUI.html` avec l'ajout des champs d'e-mail dynamiques pour le formateur et le patron.
* \* \*\*Tâches en cours :\*\*
* \* Mise en pause.
* \* \*\*Prochaines étapes planifiées :\*\*
* \* \*\*Action immédiate à la reprise :\*\* Réaliser un test de validation complet du scénario multilingue qui posait problème (répondre en anglais, retraiter en français, et inversement).
* \* Discuter et implémenter la prochaine évolution : la gestion des contenus d'e-mails et des pièces jointes en fonction des différents niveaux de restitution (`Niveau\_Details\_Resultats`, `Niveau\_Pieces\_Jointes`).
* ### 3. Contexte Historique du Projet
* Cette session marque la fin d'une phase intensive de débogage et de stabilisation. Après avoir résolu des problèmes de configuration, de nommage et de logique de base, nous nous sommes attaqués au dernier défi majeur : assurer une gestion parfaite des environnements multilingues, qui est un cas d'usage critique pour vos utilisateurs finaux. Cette étape solidifie l'architecture V2 et la rend véritablement "universelle".
* ### 4. Orientations Stratégiques
* La résolution du bug multilingue est une application directe de notre principe directeur \*\*"Le Code plus Intelligent que la Configuration"\*\*. Au lieu de dépendre de correspondances de texte fragiles, nous avons rendu le code capable de comprendre la structure sous-jacente des données (la position d'une réponse), ce qui le rend beaucoup plus résistant aux variations de contenu.
* ### 5. Architecture Technique & Ressources Clés
* #### 5.1. État des Scripts et Fichiers
* Voici les versions finales et complètes des scripts que nous avons stabilisés aujourd'hui.
* \*\*Projet `[TEMPLATE] Kit de Traitement V2` - Fichier `Utilities.gs` (Version 5.2)\*\*
* ```javascript
* // =================================================================================
* // == FICHIER : Utilities.gs - VERSION 5.2 (Correctifs finaux multilingue et destinataires)
* // == RÔLE : Boîte à outils du Kit de Traitement.
* // =================================================================================
* const ID\_FEUILLE\_PILOTE = "1kLBqIHZWbHrb4SsoSQcyVsLOmqKHkhSA4FttM5hZtDQ";
* function getTestConfiguration() {
* const ss = SpreadsheetApp.getActiveSpreadsheet();
* const idSheetActuelle = ss.getId();
* const piloteSheet = SpreadsheetApp.openById(ID\_FEUILLE\_PILOTE);
* const paramsSheet = piloteSheet.getSheetByName("Paramètres Généraux");
* if (!paramsSheet) { throw new Error("L'onglet 'Paramètres Généraux' est introuvable."); }
* const data = paramsSheet.getDataRange().getValues();
* const headers = data.shift();
* const idSheetColIndex = headers.indexOf('ID\_Sheet\_Cible');
* if (idSheetColIndex === -1) { throw new Error("La colonne 'ID\_Sheet\_Cible' est introuvable."); }
* const configRow = data.find(row => row[idSheetColIndex] === idSheetActuelle);
* if (!configRow) { throw new Error("Impossible de trouver la configuration pour ce test (ID: " + idSheetActuelle + ").");}
* const configuration = {};
* headers.forEach((header, index) => {
* if (header) { configuration[header] = configRow[index]; }
* });
* return configuration;
* }
* function getSystemIds() {
* try {
* const configSS = SpreadsheetApp.openById(ID\_FEUILLE\_PILOTE);
* const idSheet = configSS.getSheetByName('sys\_ID\_Fichiers');
* if (!idSheet) { throw new Error("L'onglet 'sys\_ID\_Fichiers' est introuvable."); }
* const data = idSheet.getDataRange().getValues();
* const ids = {};
* data.slice(1).forEach(row => {
* if (row[0] && row[1]) { ids[row[0]] = row[1]; }
* });
* return ids;
* } catch (e) {
* Logger.log("Impossible de charger les ID système : " + e.toString());
* throw new Error("Impossible de charger les ID système. Erreur: " + e.message);
* }
* }
* function getGabaritEmail(idGabarit, langueCode) {
* const systemIds = getSystemIds();
* const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);
* const gabaritsSheet = bdd.getSheetByName("Gabarits\_Emails");
* if (!gabaritsSheet) throw new Error("L'onglet 'Gabarits\_Emails' est introuvable.");
* const data = gabaritsSheet.getDataRange().getValues();
* const headers = data.shift();
* const idCol = headers.indexOf('ID\_Gabarit');
* const langCol = headers.indexOf('Langue');
* const gabaritRow = data.find(row => row[idCol] === idGabarit && row[langCol].toUpperCase() === langueCode.toUpperCase());
* if (!gabaritRow) throw new Error(`Aucun gabarit trouvé pour l'ID '${idGabarit}' et la langue '${langueCode}'.`);
* const gabarit = {};
* headers.forEach((header, index) => {
* if (header) { gabarit[header] = gabaritRow[index]; }
* });
* return gabarit;
* }
* function formatScoresDetails(resultats, niveauDetails, typeTest, langueCode) {
* if (niveauDetails === 'Simple') { return ""; }
* let scoresText = "";
* const T = loadTraductions(langueCode);
* if (niveauDetails === 'Détaillé' && resultats.scoresData) {
* if (typeTest === 'Couleurs' || typeTest === 'Ancres') {
* scoresText += (T[`INTRO\_SCORES\_DETAILLES\_${typeTest.toUpperCase()}`] || "Voici le détail de vos scores :") + "\n";
* Object.entries(resultats.scoresData).sort((a, b) => b[1] - a[1]).forEach(([code, score]) => {
* scoresText += `- ${resultats.mapCodeToName[code] || code} : ${score} ${T.SUFFIXE\_POINTS || 'points'}\n`;
* });
* } else if (typeTest === 'MBTI') {
* scoresText += (T.INTRO\_SCORES\_DETAILLES\_MBTI || "Voici le détail de vos préférences :\n");
* const s = resultats.scoresData;
* scoresText += `- ${(T.AXE\_EI || "Extraversion (E) vs Introversion (I)")} : ${s.E} vs ${s.I}\n`;
* scoresText += `- ${(T.AXE\_SN || "Sensation (S) vs Intuition (N)")} : ${s.S} vs ${s.N}\n`;
* scoresText += `- ${(T.AXE\_TF || "Pensée (T) vs Sentiment (F)")} : ${s.T} vs ${s.F}\n`;
* scoresText += `- ${(T.AXE\_JP || "Jugement (J) vs Perception (P)")} : ${s.J} vs ${s.P}\n`;
* }
* }
* return scoresText + "\n";
* }
* function buildAndSendEmails(config, reponse, resultats, langueCode, isDebugMode, destinatairesSurcharge = {}) {
* try {
* const idGabarit = config.ID\_Gabarit\_Email\_Repondant;
* if (!idGabarit) { throw new Error("La colonne 'ID\_Gabarit\_Email\_Repondant' n'est pas définie dans la configuration du test."); }
* const gabarit = getGabaritEmail(idGabarit, langueCode);
* const T = loadTraductions(langueCode);
* const variables = {
* nom\_repondant: reponse.nomRepondant || 'Participant',
* Type\_Test: config.Type\_Test || '',
* profilFinal: resultats.titreProfil || resultats.profilFinal || '',
* formateur\_nom: config.Formateur\_Nom || 'Votre Formateur',
* scoresData: formatScoresDetails(resultats, gabarit.Niveau\_Details\_Resultats, config.Type\_Test, langueCode)
* };
* let corpsEmail = `${gabarit.Corps\_Salutation || ""}\n${gabarit.Corps\_Introduction || ""}\n\n${gabarit.Corps\_Entete\_Resultats || ""}\n\n${gabarit.Corps\_Conclusion || ""}\n\n${gabarit.Corps\_Signature || ""}`;
* let sujet = gabarit.Sujet;
* for (const [key, value] of Object.entries(variables)) {
* const regex = new RegExp(`\\{${key}\\}`, 'g');
* sujet = sujet.replace(regex, value);
* corpsEmail = corpsEmail.replace(regex, value);
* }
* corpsEmail = corpsEmail.replace('\_{scoresData}\_', variables.scoresData);
* const piecesJointes = findAttachments(config.Type\_Test, resultats.profilFinal, gabarit.Niveau\_Pieces\_Jointes, langueCode);
* const adressesUniques = new Set();
* const useSurcharge = destinatairesSurcharge && Object.keys(destinatairesSurcharge).length > 0;
* if (useSurcharge) {
* if (isDebugMode) { LoggerLib.logInfo("Mode Destinataires", "Surchargé par l'UI"); }
* if (destinatairesSurcharge.repondant && reponse.emailRepondant) { adressesUniques.add(reponse.emailRepondant); }
* if (destinatairesSurcharge.formateur && destinatairesSurcharge.formateurEmail) { adressesUniques.add(destinatairesSurcharge.formateurEmail); }
* if (destinatairesSurcharge.patron && destinatairesSurcharge.patronEmail) { adressesUniques.add(destinatairesSurcharge.patronEmail); }
* if (destinatairesSurcharge.test && destinatairesSurcharge.test.trim() !== '') {
* destinatairesSurcharge.test.split(',').map(e => e.trim()).forEach(email => adressesUniques.add(email));
* }
* } else {
* if (isDebugMode) { LoggerLib.logInfo("Mode Destinataires", "Standard depuis la configuration"); }
* if (config.Repondant\_Email\_Actif === 'Oui' && reponse.emailRepondant) { adressesUniques.add(reponse.emailRepondant); }
* if (config.Patron\_Email\_Mode === 'Oui' && config.Patron\_Email) { adressesUniques.add(config.Patron\_Email); }
* if (config.Formateur\_Email\_Actif === 'Oui' && config.Formateur\_Email) { adressesUniques.add(config.Formateur\_Email); }
* }
* if (config.Developpeur\_Email) { adressesUniques.add(config.Developpeur\_Email); }
* if (isDebugMode) { LoggerLib.logEmailQueue(adressesUniques); }
* adressesUniques.forEach(adresse => {
* try {
* let sujetFinal = sujet;
* if (adresse.toLowerCase() !== (reponse.emailRepondant || "").toLowerCase()) {
* sujetFinal = (T.PREFIXE\_COPIE\_EMAIL || "Copie : ") + sujet;
* }
* if (isDebugMode) { LoggerLib.logEmailAttempt(adresse, sujetFinal); }
* MailApp.sendEmail({ to: adresse, subject: sujetFinal, body: corpsEmail, attachments: piecesJointes });
* if (isDebugMode) { LoggerLib.logSuccess(`Envoi réussi à ${adresse}`); }
* } catch (e) {
* if (isDebugMode) { LoggerLib.logError(`Échec de l'envoi à ${adresse}`, e); }
* }
* });
* if (isDebugMode) { LoggerLib.logEnd(); }
* } catch (err) {
* if (typeof LoggerLib !== 'undefined' && isDebugMode) { LoggerLib.logError("Erreur critique dans buildAndSendEmails", err); }
* Logger.log("ERREUR CRITIQUE V5.2 : " + err.toString() + "\n" + err.stack);
* }
* }
* function findAttachments(typeTest, profilCode, niveauPJ, langueCode) {
* try {
* const systemIds = getSystemIds();
* const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);
* const pjSheet = bdd.getSheetByName("sys\_PiecesJointes");
* if (!pjSheet) { return []; }
* const data = pjSheet.getDataRange().getValues();
* const headers = data.shift();
* const idx = {
* type: headers.indexOf('Type\_Test'),
* profil: headers.indexOf('Profil\_Code'),
* niveau: headers.indexOf('Email\_Niveau'),
* langue: headers.indexOf('Langue'),
* id: headers.indexOf('ID\_Fichier\_Drive')
* };
* if (Object.values(idx).some(i => i === -1)) {
* Logger.log("Avertissement : colonnes manquantes dans 'sys\_PiecesJointes'.");
* return [];
* }
* const niveauNumRequis = parseInt(niveauPJ.replace(/[^0-9]/g, ''), 10) || 1;
* const idsFichiersTrouves = new Set();
* data.forEach(row => {
* const typeMatch = (row[idx.type] === typeTest);
* const profilMatch = (row[idx.profil] === profilCode || (typeof row[idx.profil] === 'string' && row[idx.profil].toUpperCase() === 'TOUS'));
* const niveauMatch = (row[idx.niveau] > 0 && row[idx.niveau] <= niveauNumRequis);
* const langueMatch = (row[idx.langue].toUpperCase() === langueCode.toUpperCase() || row[idx.langue].toUpperCase() === 'TOUS');
* if (typeMatch && profilMatch && niveauMatch && langueMatch && row[idx.id]) {
* idsFichiersTrouves.add(row[idx.id]);
* }
* });
* if (idsFichiersTrouves.size === 0) return [];
* const fichiers = [];
* idsFichiersTrouves.forEach(id => {
* try {
* fichiers.push(DriveApp.getFileById(id).getBlob());
* } catch (e) {
* Logger.log(`Impossible d'accéder au fichier Drive avec l'ID : ${id}`);
* }
* });
* return fichiers;
* } catch (e) {
* Logger.log(`Erreur critique dans findAttachments : ${e.toString()}`);
* return [];
* }
* }
* function loadTraductions(langueCode) {
* if (!langueCode) { throw new Error("Le code de langue fourni à loadTraductions est indéfini."); }
* const systemIds = getSystemIds();
* const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);
* const traductionsSheet = bdd.getSheetByName("traductions");
* if (!traductionsSheet) throw new Error("L'onglet 'traductions' est introuvable.");
* const data = traductionsSheet.getDataRange().getValues();
* const headers = data.shift();
* const langColIndex = headers.findIndex(h => h && h.toLowerCase() === langueCode.toLowerCase());
* if (langColIndex === -1) throw new Error(`La colonne de langue '${langueCode}' est introuvable.`);
* const traductions = {};
* const keyColIndex = 0;
* data.forEach(row => {
* if (row[keyColIndex]) { traductions[row[keyColIndex]] = row[langColIndex]; }
* });
* return traductions;
* }
* function getOriginalLanguage(reponses) {
* const langueRepondantBrute = reponses['Langue / Language'] || 'Français';
* const mapLangue = { 'Français': 'FR', 'English': 'EN', 'Español': 'ES', 'Deutsch': 'DE' };
* return mapLangue[langueRepondantBrute] || 'FR';
* }
* **Projet** [TEMPLATE] Kit de Traitement V2 **- Fichier** Logique\_Universel.js **(Version 2.6)**
* // =================================================================================
* // == FICHIER : Logique\_Universel.gs
* // == VERSION : 2.6 (Correctif pour le calcul multilingue)
* // == RÔLE : Moteur de calcul universel capable de traiter n'importe quel test.
* // =================================================================================
* function calculerResultats(reponsesUtilisateur, langueCible, config, langueOrigine) {
* let resultats = {
* profilFinal: "",
* titreProfil: "",
* descriptionProfil: "",
* scoresData: {},
* sousTotauxParMode: {}
* };
* const profilsMap = \_chargerProfils(config.Type\_Test, langueCible);
* if (langueOrigine === langueCible) {
* const questionsMap = \_chargerQuestions(config.Type\_Test, langueCible);
* if (!questionsMap) return {};
* \_executerCalcul(reponsesUtilisateur, questionsMap, profilsMap, resultats, config.Type\_Test);
* } else {
* const questionsMapOrigine = \_chargerQuestions(config.Type\_Test, langueOrigine);
* const questionsMapCible = \_chargerQuestions(config.Type\_Test, langueCible);
* if (!questionsMapOrigine || !questionsMapCible) return {};
* const reponsesTraduites = {};
* for (const enTeteComplet in reponsesUtilisateur) {
* if (!enTeteComplet.includes(':')) continue;
* const idQuestion = enTeteComplet.split(':')[0].trim();
* const questionConfigOrigine = questionsMapOrigine[idQuestion];
* const questionConfigCible = questionsMapCible[idQuestion];
* if (questionConfigOrigine && questionConfigCible) {
* const reponseTexte = reponsesUtilisateur[enTeteComplet];
* const parametresOrigine = questionConfigOrigine.parametres;
* const parametresCible = questionConfigCible.parametres;
* if (parametresOrigine.options && Array.isArray(parametresOrigine.options)) {
* const reponsesArray = String(reponseTexte).split(',').map(r => r.trim());
* const reponsesSimplesTraduites = [];
* reponsesArray.forEach(reponseSimple => {
* const optionIndex = parametresOrigine.options.findIndex(opt => opt.libelle === reponseSimple);
* if (optionIndex !== -1 && parametresCible.options && parametresCible.options[optionIndex]) {
* reponsesSimplesTraduites.push(parametresCible.options[optionIndex].libelle);
* }
* });
* if(reponsesSimplesTraduites.length > 0) {
* reponsesTraduites[enTeteComplet] = reponsesSimplesTraduites.join(', ');
* }
* }
* }
* }
* \_executerCalcul(reponsesTraduites, questionsMapCible, profilsMap, resultats, config.Type\_Test);
* }
* Logger.log("Calculs terminés. Résultats : " + JSON.stringify(resultats));
* return resultats;
* }
* function \_executerCalcul(reponses, questionsMap, profilsMap, resultats, typeTest) {
* for (const enTeteComplet in reponses) {
* if (!enTeteComplet.includes(':')) continue;
* const idQuestion = enTeteComplet.split(':')[0].trim();
* const questionConfig = questionsMap[idQuestion];
* if (questionConfig) {
* const reponse = reponses[enTeteComplet];
* const mode = questionConfig.parametres.mode;
* const parametres = questionConfig.parametres;
* \_aiguillerCalcul(mode, reponse, parametres, resultats);
* }
* }
* if (Object.keys(resultats.scoresData).length > 0) {
* resultats.profilFinal = \_determinerProfilFinal(resultats.scoresData, typeTest);
* if (profilsMap[resultats.profilFinal]) {
* resultats.titreProfil = profilsMap[resultats.profilFinal].titre;
* resultats.descriptionProfil = profilsMap[resultats.profilFinal].description;
* resultats.mapCodeToName = \_creerMapCodeVersNom(profilsMap);
* }
* }
* }
* function \_aiguillerCalcul(mode, reponse, parametres, resultats) {
* switch (mode) {
* case 'QCU\_DIRECT': \_traiterQCU\_DIRECT(reponse, parametres, resultats); break;
* case 'QCU\_CAT': \_traiterQCU\_CAT(reponse, parametres, resultats); break;
* case 'QRM\_CAT': \_traiterQRM\_CAT(reponse, parametres, resultats); break;
* default: Logger.log(`Mode de traitement inconnu ou non implémenté : ${mode}`); break;
* }
* }
* function \_traiterQCU\_DIRECT(reponseUtilisateur, parametres, resultats) {
* if (!parametres || !parametres.profil) return;
* resultats.scoresData[parametres.profil] = reponseUtilisateur;
* }
* function \_traiterQCU\_CAT(reponseUtilisateur, parametres, resultats) {
* if (!parametres || !parametres.profil || !parametres.options) return;
* const optionTrouvee = parametres.options.find(opt => opt.libelle === reponseUtilisateur);
* if (optionTrouvee) {
* const valeur = (typeof optionTrouvee.valeur === 'number') ? optionTrouvee.valeur : 1;
* resultats.scoresData[parametres.profil] = (resultats.scoresData[parametres.profil] || 0) + valeur;
* }
* }
* function \_traiterQRM\_CAT(reponseUtilisateur, parametres, resultats) {
* if (!reponseUtilisateur || !parametres || !parametres.options) return;
* const reponsesArray = String(reponseUtilisateur).split(',').map(r => r.trim());
* reponsesArray.forEach(reponse => {
* const optionTrouvee = parametres.options.find(opt => opt.libelle === reponse);
* if (optionTrouvee && optionTrouvee.profil && typeof optionTrouvee.valeur === 'number') {
* const profil = optionTrouvee.profil;
* const valeur = optionTrouvee.valeur;
* resultats.scoresData[profil] = (resultats.scoresData[profil] || 0) + valeur;
* }
* });
* }
* function \_determinerProfilFinal(scoresData, typeTest) {
* if (!scoresData || Object.keys(scoresData).length === 0) return "";
* if (typeTest === 'MBTI') {
* let profil = "";
* profil += (scoresData.E || 0) > (scoresData.I || 0) ? 'E' : 'I';
* profil += (scoresData.S || 0) > (scoresData.N || 0) ? 'S' : 'N';
* profil += (scoresData.T || 0) > (scoresData.F || 0) ? 'T' : 'F';
* profil += (scoresData.J || 0) > (scoresData.P || 0) ? 'J' : 'P';
* return profil;
* } else {
* return Object.keys(scoresData).reduce((a, b) => scoresData[a] > scoresData[b] ? a : b);
* }
* }
* function \_creerMapCodeVersNom(profilsMap) {
* const map = {};
* for (const code in profilsMap) { map[code] = profilsMap[code].titre; }
* return map;
* }
* function \_chargerQuestions(typeTest, langue) {
* try {
* const systemIds = getSystemIds();
* const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);
* const nomFeuille = `Questions\_${typeTest}\_${langue}`;
* const sheet = bdd.getSheetByName(nomFeuille);
* if (!sheet) throw new Error(`Feuille introuvable: ${nomFeuille}`);
* const data = sheet.getDataRange().getValues();
* const headers = data.shift();
* const idCol = headers.indexOf('ID');
* const paramsCol = headers.indexOf('Paramètres (JSON)');
* if (idCol === -1 || paramsCol === -1) throw new Error("Colonnes ID ou 'Paramètres (JSON)' manquantes.");
* const questionsMap = {};
* data.forEach(row => {
* const id = row[idCol];
* const paramsJSON = row[paramsCol];
* if (id && paramsJSON) {
* try {
* const parametres = JSON.parse(paramsJSON);
* if (parametres.mode) { questionsMap[id] = { id: id, parametres: parametres }; }
* } catch (e) { Logger.log(`Erreur parsing JSON pour ID '${id}': ${e.message}`); }
* }
* });
* return questionsMap;
* } catch (e) {
* Logger.log("Erreur critique \_chargerQuestions: " + e.message);
* return null;
* }
* }
* function \_chargerProfils(typeTest, langue) {
* try {
* const systemIds = getSystemIds();
* const bdd = SpreadsheetApp.openById(systemIds.ID\_BDD);
* const nomFeuille = `Profils\_${typeTest}\_${langue}`;
* const sheet = bdd.getSheetByName(nomFeuille);
* if (!sheet) throw new Error(`Feuille introuvable: ${nomFeuille}`);
* const data = sheet.getDataRange().getValues();
* const headers = data.shift();
* const codeCol = 0, titreCol = 1, descCol = 2;
* const profilsMap = {};
* data.forEach(row => {
* const code = row[codeCol];
* if (code) { profilsMap[code] = { titre: row[titreCol] || "", description: row[descCol] || "" }; }
* });
* return profilsMap;
* } catch (e) {
* Logger.log("Erreur critique \_chargerProfils: " + e.message);
* return {};
* }
* }