as au même niveau d'information que maintenant

Afficher le raisonnement

Absolument. Voici la synthèse complète de notre session. Ce document contient les dernières versions de tous les scripts modifiés, y compris la correction finale pour Utilities.gs que nous venions d'identifier.

Vous pouvez le sauvegarder et me le fournir lors de notre prochaine session pour que nous reprenions exactement où nous nous sommes arrêtés.

Markdown

---

Projet: "Usine à Tests"

Date: "2025-08-07"

Version: "12.0"

---

### 1. Résumé des Échanges du Jour

La session a été une séance de débogage en profondeur et de stabilisation, principalement déclenchée par l'intégration d'un nouveau type de test, "ANCRES".

Le point de départ était un e-mail de restitution mal formaté. Notre investigation nous a menés à une refonte de la logique de génération d'e-mails, passant d'un assemblage de multiples colonnes à un modèle HTML unique dans l'onglet `Gabarits\_Emails`.

Cette modification a révélé une cascade de problèmes plus subtils que nous avons résolus un par un :

1. \*\*Préfixes `[Type Inconnu]` et `META\_` :\*\* Identifiés comme des problèmes dans le script `[MOTEUR]` qui génère les formulaires. La logique a été corrigée pour utiliser les titres lisibles et reconnaître les nouveaux types de questions.

2. \*\*Mise à jour des Scripts Dépendants :\*\* La modification des titres de questions dans les formulaires a nécessité une mise à jour du script `TraitementReponses.gs` pour qu'il puisse toujours retrouver les données du nom et de l'e-mail du répondant.

3. \*\*Calcul des Scores ANCRES :\*\* Le mode de calcul `ECHELLE\_NOTE` était manquant et a été ajouté au "moteur" `Logique\_Universel.gs`.

4. \*\*Affichage des Scores ANCRES :\*\* Le dernier problème identifié était que les scores, bien que calculés, n'apparaissaient pas dans l'e-mail final. Le diagnostic final a révélé que la fonction `formatScoresDetails` dans `Utilities.gs` ne gérait pas correctement le niveau de restitution `"le plus détaillé"` associé aux gabarits N3.

La discussion s'est conclue sur la nécessité de mettre à jour `Utilities.gs` avec ce correctif final et de valider l'ensemble de la chaîne avec un test entièrement nouveau.

### 2. État Actuel du Projet

\* \*\*Tâches réalisées :\*\*

\* Refonte complète du système de gabarits d'e-mails vers un format `Corps\_HTML` unique et robuste.

\* Correction du script `[MOTEUR]/UtilsV2.gs` pour générer correctement les questions de type `ECHELLE\_NOTE` et les questions "Méta" sans préfixes techniques.

\* Ajout du mode de calcul `ECHELLE\_NOTE` dans le script `[TEMPLATE]/Logique\_Universel.gs`.

\* Mise à jour du script `[TEMPLATE]/TraitementReponses.gs` pour correspondre aux nouveaux titres de colonnes des questions "Méta".

\* Diagnostic du bug final concernant l'affichage des scores pour les gabarits de niveau N3.

\* \*\*Tâches en cours :\*\*

\* Mise en pause de l'investigation. La solution finale pour `Utilities.gs` a été identifiée et est incluse dans ce document.

\* \*\*Prochaines étapes planifiées :\*\*

1. Remplacer le contenu du fichier `[TEMPLATE]/Utilities.gs` par la version 6.1 fournie ci-dessous.

2. Générer un \*\*tout nouveau\*\* Kit de Traitement pour le test ANCRES depuis le `[MOTEUR]`.

3. Effectuer un test de bout en bout sur ce nouveau kit pour valider la correction finale.

4. Procéder à une sauvegarde complète et isolée du système une fois celui-ci validé.

### 3. Contexte Historique du Projet

Ce projet vise à créer une "Usine à Tests" entièrement automatisée sur Google Sheets & Forms. Après une première phase de construction de l'architecture V2, nous sommes entrés dans une phase cruciale de stabilisation. La session d'aujourd'hui est caractéristique de cette phase : l'introduction d'un nouveau cas d'usage (le test ANCRES avec son mode `ECHELLE\_NOTE`) a agi comme un test de stress, révélant des faiblesses et des conventions implicites dans toute la chaîne de traitement, de la création du formulaire à l'envoi de l'e-mail final.

### 4. Orientations Stratégiques

La session a fortement renforcé le principe directeur de \*\*"Zéro Convention Implicite"\*\*. Presque chaque bug corrigé provenait d'une supposition cachée dans le code (par exemple, que le niveau de détail serait toujours "Détaillé", ou que l'ID d'une question Méta pouvait servir de titre). En rendant chaque étape plus explicite, nous augmentons la robustesse et la maintenabilité du système.

Nous avons également discuté de l'idée d'une \*\*Version 3\*\* où la logique de calcul serait définie dans la base de données plutôt que dans le code. La décision stratégique a été de \*\*reporter cette évolution majeure\*\* pour se concentrer sur la stabilisation complète de la version 2 actuelle, d'autant plus qu'une migration vers la plateforme Bubble est envisagée à long terme.

### 5. Architecture Technique & Ressources Clés

#### 5.1. État des Scripts et Fichiers :

Voici les versions complètes et à jour des 4 scripts que nous avons modifiés.

\*\*1. Fichier : `[MOTEUR]/UtilsV2.gs` (Version 2.7)\*\*

```javascript

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

// FICHIER : Utils V2.gs (Projet MOTEUR)

// RÔLE : Fonctions utilitaires pour le Moteur.

// VERSION : 2.7 - Ajout de la gestion du type de question ECHELLE\_NOTE

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

// ID de la feuille de calcul centrale qui pilote toute l'usine.

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

/\*\*

\* Lit l'onglet 'sys\_ID\_Fichiers' de la feuille de configuration

\* et retourne un objet avec tous les ID système.

\* @returns {Object} Un objet où les clés sont les noms des ID et les valeurs sont les ID.

\*/

function getSystemIds() {

try {

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

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

if (!idSheet) {

throw new Error("L'onglet de configuration 'sys\_ID\_Fichiers' est introuvable.");

}

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

const ids = {};

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

const key = row[0];

const value = row[1];

if (key && value) {

ids[key] = value;

}

});

return ids;

} catch (e) {

Logger.log("Impossible de charger les ID système : " + e.toString());

throw new Error("Impossible de charger les ID système. Vérifiez l'onglet 'sys\_ID\_Fichiers'. Erreur: " + e.message);

}

}

/\*\*

\* Récupère les données de configuration d'une ligne spécifique de manière robuste.

\* @param {number} rowIndex Le numéro de la ligne à lire.

\* @returns {Object} Un objet contenant la configuration, avec des clés correspondant exactement aux en-têtes.

\*/

function getConfigurationFromRow(rowIndex) {

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

const sheet = ss.getSheetByName('Paramètres Généraux');

if (!sheet) {

throw new Error("L'onglet 'Paramètres Généraux' est introuvable.");

}

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

const rowValues = sheet.getRange(rowIndex, 1, 1, sheet.getLastColumn()).getValues()[0];

const config = {};

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

if (header) {

config[header] = rowValues[i];

}

});

Logger.log("Configuration lue depuis la ligne " + rowIndex + " : " + JSON.stringify(config, null, 2));

return config;

}

/\*\*

\* Mappe un code langue à son nom complet.

\* @param {string} code Le code de la langue (ex: 'FR').

\* @returns {string} Le nom complet de la langue.

\*/

function getLangueFullName(code) {

const map = { 'FR': 'Français', 'EN': 'English', 'ES': 'Español', 'DE': 'Deutsch' };

return map[code.toUpperCase()] || code.toUpperCase();

}

/\*\*

\* Crée un item dans le formulaire Google en fonction de son type.

\* Gère QCU (radio), QRM (checkbox) et les différentes sources d'options (JSON/V1).

\* @param {GoogleAppsScript.Forms.Form} form L'objet formulaire auquel ajouter l'item.

\* @param {string} type Le type de question (ex: 'QCU\_CAT', 'ECHELLE\_NOTE').

\* @param {string} titre Le titre de la question.

\* @param {string} optionsString Une chaîne de caractères contenant les options, séparées par ';'.

\* @param {string} description Une chaîne de caractères pour la description / texte d'aide.

\* @param {string} paramsJSONString Une chaîne de caractères contenant les paramètres V2 au format JSON.

\*/

function creerItemFormulaire(form, type, titre, optionsString, description, paramsJSONString) {

let finalDescription = description;

const placeholderRegex = /\[LIEN\_FICHIER:(.\*?)\]/;

const match = description ? description.match(placeholderRegex) : null;

if (match && match[1]) {

const nomFichier = match[1].trim();

try {

const systemIds = getSystemIds();

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

const listeFichiersSheet = bdd.getSheetByName('Liste\_Fichiers\_Drive');

if (listeFichiersSheet) {

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

const fileRow = data.find(row => row[0].toString().trim() === nomFichier);

if (fileRow && fileRow[1]) {

const fileId = fileRow[1].toString().trim();

const fileUrl = `https://drive.google.com/file/d/${fileId}/view`;

finalDescription = description.replace(placeholderRegex, fileUrl);

} else {

finalDescription = description.replace(placeholderRegex, `[ERREUR: Fichier '${nomFichier}' introuvable dans la BDD]`);

}

} else {

finalDescription = description.replace(placeholderRegex, `[ERREUR: Onglet 'Liste\_Fichiers\_Drive' introuvable]`);

}

} catch (e) {

Logger.log("Erreur lors de la recherche du lien de fichier : " + e.message);

finalDescription = description.replace(placeholderRegex, `[ERREUR SCRIPT: ${e.message}]`);

}

}

let params = null;

let choices = [];

let item;

if (paramsJSONString && typeof paramsJSONString === 'string' && paramsJSONString.trim().startsWith('{')) {

try {

params = JSON.parse(paramsJSONString);

} catch (e) {

item = form.addParagraphTextItem().setTitle("[Erreur V2: JSON invalide] " + titre);

if(finalDescription) item.setHelpText(finalDescription);

return;

}

}

if (params && params.options && Array.isArray(params.options) && params.options.length > 0) {

choices = params.options.map(opt => (typeof opt === 'object' && opt !== null) ? opt.libelle : opt);

} else if (optionsString) {

choices = optionsString.split(';').map(String);

}

const formItemType = type ? type.toUpperCase() : '';

if (formItemType.startsWith('QRM')) {

if (choices.length > 0) {

item = form.addCheckboxItem().setTitle(titre).setChoiceValues(choices).setRequired(true);

} else {

item = form.addParagraphTextItem().setTitle("[Erreur QRM: Options manquantes] " + titre);

}

} else if (formItemType.startsWith('QCU')) {

if (choices.length > 0) {

item = form.addMultipleChoiceItem().setTitle(titre).setChoiceValues(choices).setRequired(true);

} else {

item = form.addParagraphTextItem().setTitle("[Erreur QCU: Options manquantes] " + titre);

}

} else if (formItemType === 'ECHELLE\_NOTE') {

if (params && params.echelle\_min !== undefined && params.echelle\_max !== undefined) {

const scaleItem = form.addScaleItem()

.setTitle(titre)

.setBounds(params.echelle\_min, params.echelle\_max)

.setRequired(true);

if (params.label\_min && params.label\_max) {

scaleItem.setLabels(params.label\_min, params.label\_max);

}

item = scaleItem;

} else {

item = form.addParagraphTextItem().setTitle("[Erreur ECHELLE\_NOTE: Paramètres JSON manquants] " + titre);

}

} else if (formItemType === 'ECHELLE') { // Ancien mode conservé pour compatibilité

const libelles = description ? description.split(';') : [];

if (optionsString && optionsString.split(';').length >= 2) {

const bounds = optionsString.split(';').map(Number);

const scaleItem = form.addScaleItem().setTitle(titre).setBounds(bounds[0], bounds[bounds.length - 1]).setRequired(true);

if (libelles.length === 2) {

scaleItem.setLabels(libelles[0], libelles[1]);

}

item = scaleItem;

} else {

item = form.addParagraphTextItem().setTitle("[Erreur ECHELLE: Bornes manquantes] " + titre);

}

} else if (formItemType === 'TEXTE\_EMAIL') {

const textItem = form.addTextItem().setTitle(titre).setRequired(true);

const emailValidation = FormApp.createTextValidation()

.setHelpText("Veuillez entrer une adresse e-mail valide.")

.requireTextIsEmail()

.build();

item = textItem.setValidation(emailValidation);

} else if (formItemType === 'TEXTE\_COURT') {

item = form.addTextItem().setTitle(titre).setRequired(true);

} else {

item = form.addParagraphTextItem().setTitle("[Type Inconnu: " + type + "] " + titre);

}

if (finalDescription && item && typeof item.setHelpText === 'function') {

if(formItemType !== 'ECHELLE' && formItemType !== 'ECHELLE\_NOTE') {

item.setHelpText(finalDescription);

}

}

}

**2. Fichier : [TEMPLATE]/Logique\_Universel.gs (Version 3.0)**

JavaScript

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

// == FICHIER : Logique\_Universel.gs

// == VERSION : 3.0 (Ajout du mode de calcul ECHELLE\_NOTE)

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

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

/\*\*

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

\* @param {object} reponsesUtilisateur - Les réponses lues depuis la feuille.

\* @param {string} langueCible - La langue demandée pour le résultat (ex: 'EN').

\* @param {object} config - La configuration du test.

\* @param {string} langueOrigine - La langue dans laquelle l'utilisateur a répondu (ex: 'FR').

\*/

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 {};

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());

reponsesArray.forEach(reponseSimple => {

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

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

const optionCible = parametresCible.options[optionIndex];

const reponseSimulee = {};

reponseSimulee[enTeteComplet] = optionCible.libelle;

\_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;

case 'ECHELLE\_NOTE': \_traiterECHELLE\_NOTE(reponse, parametres, resultats); break;

default:

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

break;

}

}

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

// == SOUS-FONCTIONS DE CALCUL (une par mode de traitement)

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

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;

}

});

}

/\*\*

\* Traite une réponse de type échelle de notation.

\* La réponse de l'utilisateur est directement la valeur numérique à ajouter.

\*/

function \_traiterECHELLE\_NOTE(reponseUtilisateur, parametres, resultats) {

if (!parametres || !parametres.profil) return;

const valeurNumerique = parseInt(reponseUtilisateur, 10);

if (!isNaN(valeurNumerique)) {

const profil = parametres.profil;

resultats.scoresData[profil] = (resultats.scoresData[profil] || 0) + valeurNumerique;

}

}

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

// == FONCTIONS UTILITAIRES INTERNES

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

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 {};

}

}

**3. Fichier : [TEMPLATE]/TraitementReponses.gs (Version 5.5)**

JavaScript

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

// == FICHIER : TraitementReponses.gs

// == VERSION : 5.5 (Correction des en-têtes de colonnes Méta)

// == RÔLE : Gère la logique de traitement des réponses.

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

/\*\*

\* Point d'entrée principal pour traiter une soumission de formulaire.

\*/

function onFormSubmit(e) {

try {

const rowIndex = e.range.getRow();

traiterLigne(rowIndex, {});

} catch (err) {

Logger.log(`Erreur critique dans onFormSubmit pour la ligne ${e.range.getRow()}: ${err.toString()}\n${err.stack}`);

}

}

/\*\*

\* COEUR LOGIQUE : Traite une ligne spécifique de la feuille de réponses.

\*/

function traiterLigne(rowIndex, optionsSurcharge = {}) {

const isDebugMode = (typeof LoggerLib !== 'undefined' && LoggerLib.MODE\_DEBUG\_ACTIF === true);

if (isDebugMode) { LoggerLib.logStart(`Traitement Ligne ${rowIndex}`, optionsSurcharge); }

try {

const config = getTestConfiguration();

const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheets()[0];

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

const rowValues = sheet.getRange(rowIndex, 1, 1, sheet.getLastColumn()).getValues()[0];

const reponse = {};

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

reponse[header] = rowValues[i];

});

reponse.nomRepondant = reponse['Votre nom et prénom'] || 'Participant';

reponse.emailRepondant = reponse['Votre adresse e-mail'] || null;

const langueOrigine = getOriginalLanguage(reponse);

const langueCible = optionsSurcharge.langue || langueOrigine;

if (isDebugMode) { LoggerLib.logInfo("Langue", `Origine: ${langueOrigine}, Cible: ${langueCible}`); }

const resultats = calculerResultats(reponse, langueCible, config, langueOrigine);

if (isDebugMode) { LoggerLib.logResultats(resultats); }

buildAndSendEmails(config, reponse, resultats, langueCible, isDebugMode, optionsSurcharge.destinataires);

} catch (err) {

if (isDebugMode) { LoggerLib.logError("Erreur fatale dans traiterLigne", err); }

} finally {

if (isDebugMode) { LoggerLib.logEnd(); }

}

}

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

// == SECTION INTERFACE UTILISATEUR (UI)

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

function retraiterReponse\_UI() {

const ui = SpreadsheetApp.getUi();

const result = ui.prompt(

'Retraitement de Réponse',

'Entrez le numéro de la ligne à retraiter (ex: 3) :',

ui.ButtonSet.OK\_CANCEL);

if (result.getSelectedButton() != ui.Button.OK || !result.getResponseText()) { return; }

const rowIndex = parseInt(result.getResponseText().trim(), 10);

if (isNaN(rowIndex) || rowIndex < 2) {

ui.alert('Erreur', 'Veuillez entrer un numéro de ligne valide (supérieur à 1).');

return;

}

const template = HtmlService.createTemplateFromFile('RetraitementUI');

template.ligneActive = rowIndex;

const htmlOutput = template.evaluate()

.setTitle("Retraitement - Ligne " + rowIndex)

.setWidth(350);

ui.showSidebar(htmlOutput);

}

function getDonneesPourRetraitement(rowIndex) {

try {

const config = getTestConfiguration();

const sheet = SpreadsheetApp.getActiveSpreadsheet().getSheets()[0];

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

const values = sheet.getRange(rowIndex, 1, 1, sheet.getLastColumn()).getValues()[0];

const reponse = {};

headers.forEach((h, i) => { reponse[h] = values[i]; });

return {

nomRepondant: reponse['Votre nom et prénom'] || '',

emailRepondant: reponse['Votre adresse e-mail'] || '',

langueOrigine: getOriginalLanguage(reponse),

repondantActif: config.Repondant\_Email\_Actif === 'Oui',

formateurActif: config.Formateur\_Email\_Actif === 'Oui',

patronActif: config.Patron\_Email\_Mode === 'Oui'

};

} catch (e) {

Logger.log(`ERREUR dans getDonneesPourRetraitement(${rowIndex}): ${e.toString()}`);

throw new Error("Impossible de récupérer les données pour la ligne " + rowIndex + ". " + e.message);

}

}

function lancerRetraitementDepuisUI(options) {

if (!options || !options.rowIndex) {

throw new Error("Les options de retraitement sont invalides.");

}

traiterLigne(options.rowIndex, {

langue: options.langueCible,

destinataires: options.destinataires

});

return "Retraitement pour la ligne " + options.rowIndex + " lancé avec succès !";

}

**4. Fichier : [TEMPLATE]/Utilities.gs (Version 6.1)**

JavaScript

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

// == FICHIER : Utilities.gs - VERSION 6.1 (Correction du niveau de détail "le plus détaillé")

// == RÔLE : Boîte à outils du Kit de Traitement. Contient la logique de récupération

// == des configurations, des gabarits d'e-mails et de gestion des pièces jointes.

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

// ID de la feuille de calcul centrale qui pilote toute l'usine.

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

/\*\*

\* Récupère la ligne de configuration complète pour le test en cours.

\*/

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;

}

/\*\*

\* Lit l'onglet 'sys\_ID\_Fichiers' de la feuille de configuration centrale.

\*/

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);

}

}

/\*\*

\* Récupère une ligne de gabarit d'e-mail depuis la BDD.

\*/

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;

}

/\*\*

\* Formate le bloc de scores détaillés en fonction du niveau requis.

\*/

function formatScoresDetails(resultats, niveauDetails, typeTest, langueCode) {

if (niveauDetails === 'Simple') {

return "";

}

let scoresText = "";

const T = loadTraductions(langueCode);

if (niveauDetails === 'Détaillé' || niveauDetails === 'le plus détaillé') {

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

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 || 0} vs ${s.I || 0}\n`;

scoresText += `- ${(T.AXE\_SN || "Sensation (S) vs Intuition (N)")} : ${s.S || 0} vs ${s.N || 0}\n`;

scoresText += `- ${(T.AXE\_TF || "Pensée (T) vs Sentiment (F)")} : ${s.T || 0} vs ${s.F || 0}\n`;

scoresText += `- ${(T.AXE\_JP || "Jugement (J) vs Perception (P)")} : ${s.J || 0} vs ${s.P || 0}\n`;

}

}

}

return scoresText + "\n";

}

/\*\*

\* Construit et envoie les e-mails en utilisant les gabarits de la BDD.

\*/

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 || '',

profil\_titre: resultats.titreProfil || resultats.profilFinal || '',

profil\_description: resultats.descriptionProfil || 'Aucune description disponible.',

scores\_details: formatScoresDetails(resultats, gabarit.Niveau\_Details\_Resultats, config.Type\_Test, langueCode).replace(/\n/g, '<br>'),

formateur\_nom: config.Formateur\_Nom || 'Votre Formateur',

formateur\_consultant: gabarit.formateur\_consultant || 'Votre Consultant Certifié'

};

let corpsHtml = gabarit.Corps\_HTML;

if (!corpsHtml) {

throw new Error(`Le gabarit d'e-mail '${idGabarit}' n'a pas de contenu dans la colonne 'Corps\_HTML'.`);

}

let sujet = gabarit.Sujet;

for (const [key, value] of Object.entries(variables)) {

const regex = new RegExp(`\\{${key}\\}`, 'g');

sujet = sujet.replace(regex, value);

corpsHtml = corpsHtml.replace(regex, value);

}

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 (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() !== '') {

const testEmails = destinatairesSurcharge.test.split(',').map(e => e.trim());

testEmails.forEach(email => adressesUniques.add(email));

}

} else {

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); }

adressesUniques.forEach(adresse => {

try {

let sujetFinal = sujet;

if (adresse.toLowerCase() !== (reponse.emailRepondant || "").toLowerCase()) {

sujetFinal = (T.PREFIXE\_COPIE\_EMAIL || "Copie : ") + sujet;

}

MailApp.sendEmail({

to: adresse,

subject: sujetFinal,

htmlBody: corpsHtml,

attachments: piecesJointes

});

} catch (e) {

Logger.log(`Echec de l'envoi à ${adresse}. Erreur: ${e.message}`);

}

});

} catch (err) {

Logger.log("ERREUR CRITIQUE dans buildAndSendEmails : " + err.toString() + "\n" + err.stack);

}

}

/\*\*

\* Trouve les PJ en utilisant l'en-tête 'Email\_Niveau'.

\*/

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 : une ou plusieurs colonnes sont 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 mapQuestionsById(bdd, nomFeuille) {

const sheet = bdd.getSheetByName(nomFeuille);

if (!sheet) { throw new Error(`Feuille de questions '${nomFeuille}' introuvable.`); }

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

const headers = data.shift();

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

const paramsCol = headers.indexOf('Paramètres (JSON)');

const mapById = {};

data.forEach(row => {

const qId = row[idCol];

if (qId) {

mapById[qId] = { id: qId, params: row[paramsCol] };

}

});

return mapById;

}

function getOriginalLanguage(reponses) {

const langueRepondantBrute = reponses['Langue / Language'] || 'Français';

const mapLangue = { 'Français': 'FR', 'English': 'EN', 'Español': 'ES', 'Deutsch': 'DE' };

return mapLangue[langueRepondantBrute] || 'FR';

}

**5.2. Architecture Détaillée et Traçabilité**

* **[MOTEUR]/MoteurV2.gs** : **MODIFIÉ** - La logique de création des questions "Méta" a été changée pour utiliser le titre lisible comme titre principal.
* **[MOTEUR]/UtilsV2.gs** : **MODIFIÉ** - Ajout de la logique pour créer des questions de formulaire de type ECHELLE\_NOTE à partir des paramètres JSON.
* **[TEMPLATE]/Logique\_Universel.gs** : **MODIFIÉ** - Ajout du mode de calcul ECHELLE\_NOTE pour traiter les réponses du test ANCRES.
* **[TEMPLATE]/TraitementReponses.gs** : **MODIFIÉ** - Mise à jour des en-têtes de colonnes recherchés pour le nom et l'e-mail du répondant.
* **[TEMPLATE]/Utilities.gs** : **MODIFIÉ** - Correction de la logique de la fonction formatScoresDetails pour gérer le niveau de restitution "le plus détaillé".

**5.3. Nomenclature des Variables**

* **resultats.scoresData** (Objet) : Contient les scores calculés. Origine: Logique\_Universel.gs. Exemple: {"EXPERTISE": 25, "MANAGEMENT": 30}.
* **q\_id** (String) : L'identifiant technique d'une question. Origine: Colonne ID des onglets Questions\_\*. Exemple: META\_EMAIL.
* **q\_titre** (String) : Le titre lisible d'une question. Origine: Colonne Titre des onglets Questions\_\*. Exemple: Votre adresse e-mail.
* **mode** (String) : Le mode de calcul d'une question. Origine: Paramètres (JSON). Exemple: ECHELLE\_NOTE.

**5.4. Structure des Données (Feuilles et Colonnes)**

* **Fichier : [BDD]V2 Tests & Profils**
  + **Onglet Gabarits\_Emails :** ID\_Gabarit, Langue, Sujet, Niveau\_Details\_Resultats, Niveau\_Pieces\_Jointes, Corps\_HTML.
  + **Onglet Questions\_ANCRES\_FR (exemple) :** ID, TypeQuestion, Titre, Options, Logique, Description, Paramètres (JSON).

**6. Actions Recommandées / Prochaines Étapes**

1. **Action Immédiate :** Remplacer le contenu du fichier **[TEMPLATE]/Utilities.gs** par la version 6.1 fournie dans ce document.
2. **Validation :** Générer un **tout nouveau** Kit de Traitement pour le test ANCRES.
3. **Test Final :** Soumettre une réponse à ce nouveau formulaire et vérifier que l'e-mail de restitution est 100% correct (nom du répondant, affichage des scores, etc.).
4. **Sauvegarde :** Une fois le système validé, procéder à la sauvegarde complète et isolée comme nous l'avons discuté.