**Manuel Utilisateur & Spécifications - Plateforme de Tests V5.3**

**Date de synthèse : 20 juin 2025** **Version du système : 5.3**

**PARTIE 0 : POUR LA MÉMOIRE DE GEMINI (SYNTHÈSE TECHNIQUE)**

Ce document est conçu pour servir de "sauvegarde de mémoire" complète. Si vous me le fournissez lors d'une future session, je récupérerai instantanément 100% du contexte, de l'historique, de l'architecture et des versions finales des scripts de ce projet.

* **Architecture Finale :** Le système est entièrement modulaire et repose sur deux projets Apps Script distincts :
  1. **L'Usine à Formulaires :** Un script autonome dans une feuille Google Sheet dédiée. Son unique rôle est de lire la Base de Données pour générer des couples "Formulaire + Feuille de Réponses" prêts à l'emploi. Chaque feuille de réponses est une copie du "TEMPLATE".
  2. **Le TEMPLATE - Module de Traitement :** Un projet Apps Script "modèle" contenant la logique de traitement des réponses. Ce projet est composé de plusieurs fichiers (Code.gs, CONFIGURATION.GS, LOGIQUE\_...) et est automatiquement dupliqué dans chaque nouvelle feuille de réponses générée par l'Usine.
* **Philosophie :** Le système est **piloté par les données**. Aucune modification du code n'est nécessaire pour ajouter de nouveaux tests ou de nouvelles langues. Toute la configuration se fait via la BDD et la structure des dossiers/fichiers dans Google Drive.
* **Spécificités de Construction Clés :**
  1. **Pièces Jointes Multiples :** La fonction collecterPiecesJointes est conçue pour rechercher jusqu'à 3 fichiers PDF par réponse : un fichier d'introduction général, le profil principal du participant, et un fichier complémentaire (uniquement si le niveau de réponse est "Détaillé").
  2. **Recherche Dynamique de Dossier :** La logique ne contient pas de if/else pour les types de test. Elle construit dynamiquement le nom du sous-dossier PDF à partir du type de test (ex: Pdf\_${typeTest}), rendant le système scalable.
  3. **Robustesse des Traductions :** Les fonctions de formatage des emails (formatterResultat...) utilisent une fonction "helper" t(key, fallback) qui fournit un texte de secours si une clé de traduction est manquante dans la BDD. Cela empêche le script de planter à cause d'un oubli dans le fichier de traductions.
  4. **Déclencheur Sécurisé :** La création du déclencheur (onFormSubmit) vérifie d'abord s'il en existe déjà un pour éviter les doublons et les exécutions multiples.

**PARTIE I : INTRODUCTION ET ARCHITECTURE**

Ce manuel vous guide pour maintenir et faire évoluer votre plateforme de tests de personnalité.

**1. Philosophie du Système**

La plateforme est **pilotée par les données**. Pour ajouter, modifier ou traduire un test, vous n'interviendrez **jamais** dans le code. Vous n'agirez que sur 3 éléments :

1. La **Base de Données** (une feuille Google Sheets).
2. Les **Fichiers PDF** de résultats (dans Google Drive).
3. **L'Usine à Formulaires** (pour générer les tests).

**2. Architecture de Rangement (Google Drive)**

Pour garantir la clarté et la pérennité du projet, il est vivement recommandé d'adopter la structure de rangement suivante :

📁 [PROJET] Plateforme de Tests de Personnalité

│

├─ 📁 00\_SYSTEME (Le Cœur de la Plateforme)

│ │

│ ├─ 📄 Base de Données (Google Sheet)

│ ├─ 📄 Usine à Formulaires (Google Sheet)

│ ├─ 📄 TEMPLATE - Module de Traitement (Google Sheet)

│ └─ 📁 Dossier PDF Base

│ │

│ ├─ 📁 Pdf\_Couleurs

│ ├─ 📁 Pdf\_MBTI

│ └─ ... (un dossier par type de test)

│

└─ 📁 01\_TESTS\_GENERES (Les Fichiers de Production)

│

├─ 📁 Formulaires (Google Forms)

└─ 📁 Feuilles de Réponses (Google Sheets)

**3. Composants de la Plateforme**

Votre système repose sur 4 briques logicielles situées dans le dossier 00\_SYSTEME.

* **📄 Base de Données :** Le cerveau de l'application. Contient les questions, les profils et les traductions de tous les tests.
* **🏭 Usine à Formulaires :** Votre interface pour créer de nouveaux questionnaires. C'est le seul outil que vous manipulerez activement.
* **⚙️ TEMPLATE - Module de Traitement :** Le "moteur" de traitement qui est automatiquement copié dans chaque nouveau test. Vous ne le modifiez que si nous décidons d'une mise à jour majeure du système.
* **📁 Dossier PDF Base :** Le dossier qui contient tous les fichiers PDF de résultats, organisés par sous-dossiers.

**PARTIE II : AJOUTER UN NOUVEAU TEST**

Suivez ces 4 étapes pour ajouter un nouveau test (ex: "DISC").

**Étape 1 : Préparer la Base de Données**

1. **Ouvrez la feuille Base de Données**.
2. **Créez les onglets de Questions :**
   * Créez un onglet nommé Questions\_DISC\_fr.
   * Créez un onglet nommé Questions\_DISC\_en.
   * Remplissez les colonnes (Type, Titre, Options, Logique, etc.).
3. **Créez les onglets de Profils :**
   * Créez un onglet nommé Profils\_DISC\_fr.
   * Créez un onglet nommé Profils\_DISC\_en.
   * Remplissez les colonnes (Type, Nom Complet, Titre Profil, Description Complète).

**POINT CRUCIAL :** La valeur dans la colonne Logique (Questions) doit correspondre **exactement** à la valeur dans la colonne Type (Profils). D est différent de d.

**Étape 2 : Préparer les Fichiers PDF**

1. **Naviguez vers le dossier Dossier PDF Base**.
2. **Créez le sous-dossier de Test :**
   * Créez un nouveau sous-dossier. Son nom doit **impérativement** suivre la nomenclature Pdf\_[NomDuTest].
   * Pour notre exemple, créez le dossier **Pdf\_DISC**.
3. **Ajoutez et nommez les fichiers PDF** à l'intérieur de Pdf\_DISC :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type de Fichier** | **Rôle** | **Nomenclature Requise** | **Exemple** | **Obligatoire ?** |
| **Introduction** | Document général sur le test. | [NomDuTest]\_[langue]\_introduction.pdf | DISC\_fr\_introduction.pdf | Non |
| **Profil Principal** | Description du profil du participant. | [CodeProfil]\_[langue].pdf | D\_fr.pdf | **Oui** |
| **Complément** | Infos additionnelles (mode "Détaillé"). | [CodeProfil]\_[langue]\_2.pdf | D\_fr\_2.pdf | Non |

**LES RÈGLES D'OR DE LA NOMENCLATURE :**

1. **La Casse :** DISC et disc sont différents. Soyez cohérent partout.
2. **Le Séparateur :** Utilisez toujours le tiret bas \_. Jamais d'espaces ni de tirets.
3. **Les Noms :** [NomDuTest] (ex: DISC) et [CodeProfil] (ex: D) doivent être identiques à ceux de votre BDD.
4. **L'Extension :** Toujours terminer par .pdf.

**Étape 3 : Générer le Questionnaire**

1. **Ouvrez la feuille Usine à Formulaires**.
2. Lancez la génération via le menu **🏭 Usine à Tests (Automatisée) > 1. Générer un nouveau Questionnaire**.
3. Suivez les 5 étapes du formulaire (choix du test, nombre de questions, niveau de réponse, etc.).
4. Une fois la génération terminée, **rangez manuellement** le nouveau Formulaire et la nouvelle Feuille de Réponses dans les sous-dossiers correspondants de 01\_TESTS\_GENERES.

**Étape 4 : Activer et Valider le Test**

1. Ouvrez la nouvelle feuille de réponses qui vient d'être créée.
2. Cliquez sur le menu **⚙️ Configuration Test > 1. Activer le Déclencheur**. Autorisez le script si c'est la première fois.
3. Répondez vous-même au formulaire pour tester et valider que vous recevez bien l'email avec les bonnes pièces jointes.

**PARTIE III : AJOUTER UNE NOUVELLE LANGUE**

Suivez ces 3 étapes pour ajouter une nouvelle langue (ex: l'Espagnol, es).

1. **Dans la Base de Données :**
   * Créez les onglets Questions\_[NomDuTest]\_es et Profils\_[NomDuTest]\_es pour chaque test.
   * Ouvrez l'onglet Traductions et ajoutez une nouvelle colonne avec l'en-tête es. Remplissez toutes les traductions.
2. **Dans les Dossiers PDF :**
   * Pour chaque test, ajoutez la version espagnole de chaque PDF en respectant la nomenclature.
   * Exemples : DISC\_es\_introduction.pdf, D\_es.pdf, D\_es\_2.pdf.
3. **Dans l'Usine à Formulaires :**
   * Aucune action requise. L'Usine détectera automatiquement la nouvelle langue si les onglets correspondants existent dans la BDD.

**PARTIE IV : MAINTENANCE ET POINTS D'ATTENTION**

**Mise à jour du "Moteur"**

Si nous décidons d'améliorer le système, la seule action requise sera de mettre à jour les scripts contenus dans le projet **TEMPLATE - Module de Traitement**. Seuls les tests générés *après* la mise à jour bénéficieront des améliorations.

**⚠️ Script Obsolète dans la Base de Données**

Lors de nos analyses, nous avons identifié un script très ancien directement attaché à votre fichier Base de Données. Ce script est une version de développement primitive et n'est plus utilisé. **Action requise :**

1. Ouvrez le fichier Base de Données.
2. Allez dans Extensions > Apps Script.
3. Supprimez l'intégralité du code qui s'y trouve.
4. Enregistrez le projet vide.

*Ceci est impératif pour éviter tout conflit ou exécution inattendue.*

**PARTIE V : ANNEXE - SCRIPTS COMPLETS (VERSION 5.3)**

Voici l'intégralité des scripts à jour qui composent votre plateforme.

**1. Script de "l'Usine à Formulaires"**

*(À placer dans le projet Apps Script de la feuille "Usine à Formulaires")*

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

// == SCRIPT DE L'USINE À FORMULAIRES V5.3 - VERSION FINALE STABILISÉE

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

// --- IDs et Configuration Globale ---

const ID\_BDD = "1m2MGBd0nyiAl3qw032B6Nfj7zQL27bRSBexiOPaRZd8";

const ID\_DOSSIER\_PDF\_BASE = "1Rgj1RvJybJgQ-0Dxp3Qv3Jf3XuhlcVng";

const ID\_TEMPLATE\_REPONSES = "1kCAotNmrTPBDprMxdYOMjHvcy33r6qloKqS6\_BYjDAY";

/\*\*

\* Crée le menu personnalisé dans l'interface utilisateur de Google Sheets.

\*/

function onOpen() {

SpreadsheetApp.getUi()

.createMenu(' 🏭 Usine à Tests (Automatisée)')

.addItem("1. Générer un nouveau Questionnaire", "lancerCreationFormulaire")

.addToUi();

}

/\*\*

\* Mappe un code langue à son nom complet.

\*/

function getLangueFullName(code) {

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

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

}

/\*\*

\* Fonction principale qui orchestre la création d'un test.

\*/

function lancerCreationFormulaire() {

const ui = SpreadsheetApp.getUi();

try {

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

const toutesLesFeuilles = bdd.getSheets();

const regexTests = /^Questions\_([a-zA-Z0-9]+)\_[A-Z]{2}$/i;

const testsDisponibles = {};

toutesLesFeuilles.forEach(feuille => {

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

if (match && match[1]) { testsDisponibles[match[1]] = true; }

});

const nomTestsDisponibles = Object.keys(testsDisponibles).sort();

if (nomTestsDisponibles.length === 0) throw new Error("Aucun type de test valide détecté dans la Base de Données.");

let promptTestText = 'Tests disponibles détectés :\n\n';

nomTestsDisponibles.forEach((nom, index) => {

promptTestText += `${index + 1}. ${nom}\n`;

});

promptTestText += '\nEntrez le numéro du test à générer.';

const testResponse = ui.prompt('Étape 1/5 : Choix du Test', promptTestText, ui.ButtonSet.OK\_CANCEL);

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

const numeroTestChoisi = parseInt(testResponse.getResponseText().trim(), 10);

if (isNaN(numeroTestChoisi) || numeroTestChoisi < 1 || numeroTestChoisi > nomTestsDisponibles.length) {

throw new Error(`Numéro de test invalide.`);

}

const nomTest = nomTestsDisponibles[numeroTestChoisi - 1];

const regexLangues = new RegExp(`^Questions\_${nomTest}\_([A-Z]{2})$`);

const languesAInclure = [];

toutesLesFeuilles.forEach(feuille => {

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

if (match) {

const code = match[1];

languesAInclure.push({ code: code.toLowerCase(), nomComplet: getLangueFullName(code), feuille: feuille });

}

});

if (languesAInclure.length === 0) throw new Error(`Aucune langue trouvée pour le test "${nomTest}".`);

languesAInclure.sort((a, b) => a.code.localeCompare(b.code));

const ordreLanguesString = languesAInclure.map(l => l.code).join(',');

const nbQuestionsMax = languesAInclure[0].feuille.getLastRow() - 1;

const nbQuestionsResponse = ui.prompt(`Étape 2/5 : Nombre de Questions`, `Maximum : ${nbQuestionsMax}.\n\nCombien de questions souhaitez-vous inclure ?`, ui.ButtonSet.OK\_CANCEL);

if (nbQuestionsResponse.getSelectedButton() !== ui.Button.OK) { return; }

const nbQuestionsChoisi = parseInt(nbQuestionsResponse.getResponseText(), 10);

if (isNaN(nbQuestionsChoisi) || nbQuestionsChoisi < 1 || nbQuestionsChoisi > nbQuestionsMax) { throw new Error(`Valeur invalide.`); }

const niveauResponse = ui.prompt("Étape 3/5 : Niveau de Réponse", "Choisissez le niveau de restitution pour l'email final :\n\nEntrez '1' pour Simple.\nEntrez '2' pour Détaillé.", ui.ButtonSet.OK\_CANCEL);

const reponseNum = niveauResponse.getResponseText().trim();

let niveauReponseChoisi = "";

if (reponseNum === '1') { niveauReponseChoisi = 'Simple'; }

else if (reponseNum === '2') { niveauReponseChoisi = 'Détaillé'; }

else { throw new Error("Réponse invalide. Veuillez entrer '1' ou '2'."); }

const adminEmailResponse = ui.prompt("Étape 4/5 : Copie Administrateur (Optionnel)","Entrez une adresse e-mail si une personne doit recevoir une copie de chaque résultat.\n\nLaissez vide sinon.", ui.ButtonSet.OK\_CANCEL);

if (adminEmailResponse.getSelectedButton() !== ui.Button.OK) { return; }

const emailAdmin = adminEmailResponse.getResponseText().trim();

const statutResponse = ui.prompt("Étape 5/5 : Statut du Test", "Entrez le statut du test (ex: 'PROD', 'TEST').", ui.ButtonSet.OK\_CANCEL);

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

const statutFinal = statutResponse.getResponseText().trim().toUpperCase();

const titreInterne = `[${statutFinal}] Questionnaire ${nomTest} (${nbQuestionsChoisi} Questions)`;

const titrePublic = `Test de Personnalité / Personality Test - ${nomTest}`;

const templateFile = DriveApp.getFileById(ID\_TEMPLATE\_REPONSES);

const newSheetFile = templateFile.makeCopy().setName(`Réponses - ${titreInterne}`);

const ssDestination = SpreadsheetApp.openById(newSheetFile.getId());

const form = FormApp.create(titreInterne).setTitle(titrePublic);

form.setDestination(FormApp.DestinationType.SPREADSHEET, ssDestination.getId());

form.setProgressBar(true);

form.addTextItem().setTitle("Nom Complet / Full Name").setRequired(true);

form.addTextItem().setTitle("Adresse Email").setValidation(FormApp.createTextValidation().requireTextIsEmail().build()).setRequired(true);

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 questionsData = langue.feuille.getRange(2, 1, nbQuestionsChoisi, 6).getValues();

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

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

creerItemFormulaire(form, type, `${id}: ${titre}`, options, description);

if (index === questionsData.length - 1) page.setGoToPage(FormApp.PageNavigationType.SUBMIT);

});

});

itemLangue.setChoices(choices);

let configSheet = ssDestination.getSheetByName('\_Config');

if (!configSheet) { configSheet = ssDestination.insertSheet('\_Config'); }

configSheet.clear();

configSheet.getRange('A1:B7').setValues([

['TYPE\_TEST', nomTest],

['ID\_BDD', ID\_BDD],

['ID\_DOSSIER\_PDF', ID\_DOSSIER\_PDF\_BASE],

['NIVEAU\_REPONSE', niveauReponseChoisi],

['ORDRE\_LANGUES', ordreLanguesString],

['EMAIL\_ADMIN\_FORMULAIRE', emailAdmin],

['NB\_QUESTIONS', nbQuestionsChoisi]

]);

configSheet.hideSheet();

ui.alert(

`SUCCÈS !\n\nLe test "${titreInterne}" a été généré.\n\n` +

`Lien de partage : \n${form.getPublishedUrl()}\n\n` +

" ❗ ACTION MANUELLE REQUISE :\n" +

"1. Ouvrez la nouvelle feuille de réponses qui vient d'être créée.\n" +

"2. Allez dans son menu \" ⚙️ Configuration Test > 1. Activer le Déclencheur\" pour que les réponses soient traitées."

);

} catch (e) {

ui.alert(`ERREUR : ${e.message}`);

}

}

/\*\*

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

\*/

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

const options = optionsString ? optionsString.split(';') : [];

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

switch (type.toUpperCase()) {

case 'CHOIX\_BINAIRE':

form.addMultipleChoiceItem().setTitle(titre).setChoiceValues(options).setRequired(true);

break;

case 'CHOIX\_MULTIPLE':

form.addCheckboxItem().setTitle(titre).setChoiceValues(options).setRequired(true);

break;

case 'ECHELLE':

const scaleItem = form.addScaleItem().setTitle(titre).setBounds(parseInt(options[0], 10), parseInt(options[1], 10)).setRequired(true);

if (libelles.length === 2) { scaleItem.setLabels(libelles[0], libelles[1]); }

break;

case 'CLASSEMENT':

const gridItem = form.addGridItem().setTitle(titre).setHelpText("Classez chaque option.").setRows(options).setRequired(true);

const colonnesClassement = options.map((opt, i) => `Rang ${i + 1}`);

gridItem.setColumns(colonnesClassement);

break;

default:

form.addParagraphTextItem().setTitle(`[Type Inconnu: ${type}] ${titre}`);

break;

}

}

**2. Scripts du "TEMPLATE - Module de Traitement"**

*(Ces 4 fichiers cohabitent dans le même projet Apps Script)*

**Fichier 1 : Code.gs**

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

// == SCRIPT DE TRAITEMENT PRINCIPAL (CODE.GS) - V5.3 PIECES JOINTES MULTIPLES

// == Logique centrale qui appelle les fonctions de support externes.

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

/\*\*

 \* Fonction principale déclenchée par la soumission d'un formulaire.

 \* Point d'entrée unique du traitement.

 \*/

function traitementDesReponses(e) {

  const logMessages = [];

  const log = (message) => logMessages.push(`[${new Date().toLocaleTimeString()}] ${message}`);

  let status = "ERREUR";

  let respondentEmail = (e && e.values && e.values.length > 2) ? e.values[2] : "Inconnu";

  try {

    log("--- Début du traitement (V5.3) ---");

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    const configSheet = ss.getSheetByName('\_Config');

    if (!configSheet) throw new Error("Onglet de configuration '\_Config' introuvable.");

    const configData = configSheet.getRange('A1:B7').getValues();

    const config = configData.reduce((obj, row) => { if(row[0] && row[1] !== undefined) obj[row[0]] = row[1]; return obj; }, {});

    if (!config.NB\_QUESTIONS) throw new Error("NB\_QUESTIONS non défini. Veuillez régénérer le formulaire avec la dernière version de l'Usine.");

    log(`Config lue. Nb questions: ${config.NB\_QUESTIONS}, Niveau: ${config.NIVEAU\_REPONSE}`);

    const reponses = e.values;

    const nom = reponses[1];

    const email = reponses[2];

    const langue = reponses[3];

    const codeLangue = langue.substring(0, 2).toLowerCase();

    log(`Infos: Nom=${nom}, Email=${email}, Langue=${langue} (${codeLangue})`);

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

    log("Connexion BDD ok.");

    const trads = chargerTraductions(bdd, codeLangue);

    log(`Traductions chargées pour '${codeLangue}'.`);

    let reponsesUtilisateur;

    const ordreLangues = config.ORDRE\_LANGUES ? config.ORDRE\_LANGUES.split(',') : [];

    const indexLangueChoisie = ordreLangues.indexOf(codeLangue);

    let questionsASauter = 0;

    if (indexLangueChoisie > 0) {

      const nombreDeQuestionsParBloc = parseInt(config.NB\_QUESTIONS, 10);

      questionsASauter = indexLangueChoisie \* nombreDeQuestionsParBloc;

    }

    reponsesUtilisateur = reponses.slice(4 + questionsASauter);

    log(`Décalage CORRIGÉ: ${questionsASauter} questions sautées.`);

    let scores = {};

    const nomFeuilleQuestions = `Questions\_${config.TYPE\_TEST}\_${codeLangue.toUpperCase()}`;

    const feuilleQuestions = bdd.getSheetByName(nomFeuilleQuestions);

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

    const questions = feuilleQuestions.getRange(2, 1, config.NB\_QUESTIONS, 6).getValues();

    let reponseIndex = 0;

    questions.forEach((q\_data) => {

      if (reponseIndex >= reponsesUtilisateur.length) return;

      const [id, type, titre, optionsStr, logiqueStr, description] = q\_data;

      if (!logiqueStr) { reponseIndex++; return; }

      const logiques = logiqueStr.split(';');

      switch (type.toUpperCase()) {

        case 'CHOIX\_BINAIRE': {

          const options = optionsStr.split(';');

          const indexChoix = options.indexOf(reponsesUtilisateur[reponseIndex]);

          if (indexChoix !== -1) { scores[logiques[indexChoix]] = (scores[logiques[indexChoix]] || 0) + 1; }

          reponseIndex++; break;

        }

        case 'CHOIX\_MULTIPLE': {

          const options = optionsStr.split(';');

          const reponsesCochees = reponsesUtilisateur[reponseIndex] ? reponsesUtilisateur[reponseIndex].split(', ') : [];

          reponsesCochees.forEach(reponse => {

            const indexChoix = options.indexOf(reponse);

            if (indexChoix !== -1) { scores[logiques[indexChoix]] = (scores[logiques[indexChoix]] || 0) + 1; }

          });

          reponseIndex++; break;

        }

        case 'ECHELLE': {

          scores[logiques[0]] = (scores[logiques[0]] || 0) + parseInt(reponsesUtilisateur[reponseIndex], 10);

          reponseIndex++; break;

        }

        case 'CLASSEMENT': {

          const options = optionsStr.split(';');

          for (let i = 0; i < options.length; i++) {

            const reponseGrille = reponsesUtilisateur[reponseIndex + i];

            if (!reponseGrille) continue;

            const rang = parseInt(reponseGrille.replace('Rang ', ''), 10);

            const points = options.length - rang + 1;

            scores[logiques[i]] = (scores[logiques[i]] || 0) + points;

          }

          reponseIndex += options.length; break;

        }

        default: reponseIndex++; break;

      }

    });

    log(`Scores calculés: ${JSON.stringify(scores)}`);

    const nomFeuilleProfils = `Profils\_${config.TYPE\_TEST}\_${codeLangue.toUpperCase()}`;

    const feuilleProfils = bdd.getSheetByName(nomFeuilleProfils);

    if (!feuilleProfils) throw new Error(`Feuille profils '${nomFeuilleProfils}' introuvable.`);

    const profilsData = feuilleProfils.getRange(2, 1, feuilleProfils.getLastRow() - 1, 4).getValues();

    let resultat;

    if (config.TYPE\_TEST.toUpperCase() === 'MBTI') {

      resultat = formatterResultatMBTI(scores, profilsData, nom, config.NIVEAU\_REPONSE, trads);

    } else if (config.TYPE\_TEST.toUpperCase() === 'COULEURS') {

      resultat = formatterResultatCouleurs(scores, profilsData, nom, config.NIVEAU\_REPONSE, trads);

    } else {

      throw new Error(`Type de test '${config.TYPE\_TEST}' non supporté.`);

    }

    if (!resultat || !resultat.profilFinal) throw new Error("Détermination du profil impossible.");

    log(`Profil final: ${resultat.profilFinal}.`);

    if (resultat && resultat.corpsEmail) {

      const sujetEmail = `Vos résultats pour le test ${config.TYPE\_TEST}`;

      const corpsHtml = resultat.corpsEmail.replace(/\n/g, '<br>');

      const piecesJointes = collecterPiecesJointes(resultat.profilFinal, codeLangue, config.ID\_DOSSIER\_PDF, config.TYPE\_TEST, config.NIVEAU\_REPONSE, log);

      const bccList = ["chanenam@gmail.com"];

      if (config.EMAIL\_ADMIN\_FORMULAIRE && config.EMAIL\_ADMIN\_FORMULAIRE.trim() !== "") { bccList.push(config.EMAIL\_ADMIN\_FORMULAIRE.trim()); }

      const mailOptions = { to: email, subject: sujetEmail, htmlBody: corpsHtml, attachments: piecesJointes, bcc: bccList.join(',') };

      MailApp.sendEmail(mailOptions);

      log(`Email envoyé à ${email}. Copie(s) à : ${mailOptions.bcc}. ${piecesJointes.length} PJ(s) jointe(s).`);

      status = "SUCCÈS";

    } else { throw new Error("Corps de l'email vide."); }

  } catch (err) {

    log(`ERREUR CRITIQUE : ${err.toString()}\n${err.stack}`);

    status = "ERREUR";

  } finally {

    const ss = SpreadsheetApp.getActiveSpreadsheet();

    let logSheet = ss.getSheetByName('Log');

    if (!logSheet) {

      logSheet = ss.insertSheet('Log');

      logSheet.getRange('A1:D1').setValues([['Horodatage', 'Email Répondant', 'Statut', 'Détails des Événements']]).setFontWeight('bold');

      logSheet.setColumnWidth(1, 150).setColumnWidth(2, 200).setColumnWidth(3, 100).setColumnWidth(4, 500);

    }

    logSheet.insertRowAfter(1);

    logSheet.getRange('A2:D2').setValues([[new Date(), respondentEmail, status, logMessages.join('\n')]]);

    logSheet.getRange('A2:D2').setWrap(true);

  }

}

function chargerTraductions(bdd, codeLangue) {

  const feuilleTrads = bdd.getSheetByName('Traductions');

  if (!feuilleTrads) throw new Error("Onglet 'Traductions' introuvable.");

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

  const langColIndex = headers.indexOf(codeLangue);

  if (langColIndex === -1) throw new Error(`Code langue "${codeLangue}" introuvable dans les en-têtes [${headers.join(',')}]`);

  const data = feuilleTrads.getRange(2, 1, feuilleTrads.getLastRow() - 1, feuilleTrads.getLastColumn()).getValues();

  return data.reduce((obj, row) => {

    const key = row[0] ? row[0].trim() : '';

    if (key) { obj[key] = row[langColIndex]; }

    return obj;

  }, {});

}

function collecterPiecesJointes(profilFinal, codeLangue, idDossierBase, typeTest, niveauReponse, log) {

  let piecesJointes = [];

  if (!idDossierBase) {

    log("Info: ID dossier PJ non configuré.");

    return piecesJointes;

  }

  try {

    const dossierBase = DriveApp.getFolderById(idDossierBase);

const nomSousDossier = `Pdf\_${typeTest}`;

log(`Recherche des pièces jointes dans le sous-dossier : '${nomSousDossier}'...`);

const sousDossiers = dossierBase.getFoldersByName(nomSousDossier);

if (!sousDossiers.hasNext()) {

log(`ERREUR: Le sous-dossier requis '${nomSousDossier}' est introuvable.`);

return piecesJointes;

}

const dossierCible = sousDossiers.next();

const nomFichierIntro = `${typeTest}\_${codeLangue}\_introduction.pdf`;

const fichiersIntro = dossierCible.getFilesByName(nomFichierIntro);

if (fichiersIntro.hasNext()) {

piecesJointes.push(fichiersIntro.next().getBlob());

log(`PJ trouvée : '${nomFichierIntro}'`);

}

const nomFichierProfil = `${profilFinal}\_${codeLangue}.pdf`;

const fichiersProfil = dossierCible.getFilesByName(nomFichierProfil);

if (fichiersProfil.hasNext()) {

piecesJointes.push(fichiersProfil.next().getBlob());

log(`PJ trouvée : '${nomFichierProfil}'`);

} else {

log(`Avertissement: PJ principale '${nomFichierProfil}' non trouvée.`);

}

if (niveauReponse && niveauReponse.toUpperCase() === 'DÉTAILLÉ') {

const nomFichierComp = `${profilFinal}\_${codeLangue}\_2.pdf`;

const fichiersComp = dossierCible.getFilesByName(nomFichierComp);

if (fichiersComp.hasNext()) {

piecesJointes.push(fichiersComp.next().getBlob());

log(`PJ trouvée (mode Détaillé) : '${nomFichierComp}'`);

}

}

  } catch (e) {

    log(`Erreur collecte PJ: ${e.toString()}`);

  }

  return piecesJointes;

}

**Fichier 2 : CONFIGURATION.GS**

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

// == FICHIER : CONFIGURATION.GS

// == Contient les fonctions d'initialisation et le menu de configuration.

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

function onOpen() {

  SpreadsheetApp.getUi()

    .createMenu('⚙️ Configuration Test')

    .addItem("1. Activer le Déclencheur", "creerDeclencheurSiNecessaire")

    .addToUi();

}

function creerDeclencheurSiNecessaire() {

  const ss = SpreadsheetApp.getActive();

  const declencheurs = ScriptApp.getUserTriggers(ss);

  const declencheurExiste = declencheurs.some(d => d.getHandlerFunction() === 'traitementDesReponses');

  if (!declencheurExiste) {

    ScriptApp.newTrigger('traitementDesReponses')

      .forSpreadsheet(ss)

      .onFormSubmit()

      .create();

    SpreadsheetApp.getUi().alert('✅ Le déclencheur a été activé avec succès ! Cette feuille est prête.');

  } else {

    SpreadsheetApp.getUi().alert('ℹ️ Le déclencheur est déjà actif sur cette feuille. Aucune action n\'a été nécessaire.');

  }

}

**Fichier 3 : LOGIQUE\_COULEURS.GS**

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

// == FICHIER : LOGIQUE\_COULEURS.GS (V1.2 - Robuste)

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

function formatterResultatCouleurs(scores, profilsData, nom, niveauReponse, trads) {

  const t = (key, fallback) => trads[key] || fallback;

  const profilFinal = Object.keys(scores).reduce((a, b) => (scores[a] || 0) > (scores[b] || 0) ? a : b, '');

  const profilInfo = profilsData.find(p => p[0] === profilFinal);

  const [code, nomCompletDom, titreDom, descDom] = profilInfo ? profilInfo : [profilFinal, profilFinal, "Profil Inconnu", ""];

  const salutation = t('SALUTATION', 'Bonjour {name},').replace('{name}', nom);

  let corpsEmail = "";

  if (niveauReponse.toUpperCase() === 'DÉTAILLÉ') {

    const mapNomsComplets = profilsData.reduce((map, row) => { map[row[0]] = row[1]; return map; }, {});

    let detailsScores = t('INTRO\_SCORES\_DETAILLES\_COULEURS', 'Voici le détail de vos scores :') + "\n";

    const scoresTries = Object.entries(scores).sort((a, b) => b[1] - a[1]);

    scoresTries.forEach(([couleur, score]) => {

      const nomComplet = mapNomsComplets[couleur] || couleur;

      detailsScores += `- ${nomComplet} : ${score} ${t('SUFFIXE\_POINTS', 'points')}\n`;

    });

    const conclusion = t('CONCLUSION\_DOMINANTE\_COULEURS', 'Votre couleur dominante est le {nomCouleur}')

        .replace('{nomCouleur}', `\*\*${nomCompletDom} (${titreDom})\*\*`);

    corpsEmail = `${salutation}\n\n${detailsScores}\n${conclusion}.\n\n${descDom}`;

  } else {

    const conclusion = t('CONCLUSION\_COULEURS\_SIMPLE', 'Votre profil dominant est le {nomCouleur}')

        .replace('{nomCouleur}', `\*\*${nomCompletDom} (${titreDom})\*\*`);

    corpsEmail = `${salutation}\n\n${conclusion}.\n\n${descDom}`;

  }

  return { profilFinal: profilFinal, corpsEmail: corpsEmail };

}

**Fichier 4 : LOGIQUE\_MBTI.GS**

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

// == FICHIER : LOGIQUE\_MBTI.GS (V1.2 - Robuste)

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

function formatterResultatMBTI(scores, profilsData, nom, niveauReponse, trads) {

  const t = (key, fallback) => trads[key] || fallback;

  const profilFinal =

    ((scores.E || 0) >= (scores.I || 0) ? 'E' : 'I') +

    ((scores.S || 0) >= (scores.N || 0) ? 'S' : 'N') +

    ((scores.T || 0) >= (scores.F || 0) ? 'T' : 'F') +

    ((scores.J || 0) >= (scores.P || 0) ? 'J' : 'P');

  const profilInfo = profilsData.find(p => p[0] === profilFinal);

  const [code, nomComplet, titre, description] = profilInfo ? profilInfo : [profilFinal, profilFinal, "Profil Inconnu", ""];

  const salutation = t('SALUTATION', 'Bonjour {name},').replace('{name}', nom);

  const conclusion = `${t('CONCLUSION\_MBTI\_SIMPLE', 'Votre type de personnalité est :')} \*\*${profilFinal} – ${titre}\*\*`;

  let corpsEmail = "";

  if (niveauReponse.toUpperCase() === 'DÉTAILLÉ') {

    let detailsScores = t('INTRO\_SCORES\_DETAILLES\_MBTI', 'Voici le détail de vos préférences :') + "\n";

    detailsScores += "\* " + t('MBTI\_E\_I', 'Extraversion (E) vs Introversion (I)').replace('{score1}', scores.E || 0).replace('{score2}', scores.I || 0) + "\n";

    detailsScores += "\* " + t('MBTI\_S\_N', 'Sensation (S) vs Intuition (N)').replace('{score1}', scores.S || 0).replace('{score2}', scores.N || 0) + "\n";

    detailsScores += "\* " + t('MBTI\_T\_F', 'Pensée (T) vs Sentiment (F)').replace('{score1}', scores.T || 0).replace('{score2}', scores.F || 0) + "\n";

    detailsScores += "\* " + t('MBTI\_J\_P', 'Jugement (J) vs Perception (P)').replace('{score1}', scores.J || 0).replace('{score2}', scores.P || 0) + "\n";

    corpsEmail = `${salutation}\n\n${detailsScores}\n${conclusion}\n\n${description}`;

  } else {

    corpsEmail = `${salutation}\n\n${conclusion}\n\n${description}`;

  }

  return { profilFinal: profilFinal, corpsEmail: corpsEmail };

}