Voici le document final et complet, prêt à être sauvegardé.

Markdown

---

Projet: "Usine à Tests"

Date: "2508211930"

Version: "17.0"

---

### 1. Résumé des Échanges du Jour

La session a été entièrement consacrée au développement de la première brique des "Évolutions E-mail" : l'\*\*envoi d'un e-mail de confirmation immédiat\*\* au répondant. Ce qui semblait être une simple fonctionnalité a révélé plusieurs points de friction que nous avons résolus méthodiquement :

1. \*\*Implémentation Initiale\*\* : Nous avons d'abord implémenté la fonctionnalité en utilisant un Google Doc comme gabarit, piloté par une nouvelle colonne dans `Paramètres Généraux`.

2. \*\*Gestion Multilingue\*\* : Nous avons immédiatement identifié la nécessité de gérer plusieurs langues, ce qui a fait évoluer la configuration vers des colonnes spécifiques par langue (ex: `ID\_Gabarit\_Email\_Confirmation\_FR`).

3. \*\*Personnalisation\*\* : L'e-mail a été rendu plus personnel en y intégrant la logique de remplacement des placeholders (ex: `{{Votre\_nom\_et\_prenom}}`).

4. \*\*Optimisation de la Configuration\*\* : Pour éviter la redondance, nous avons adopté une approche plus robuste : les ID des gabarits standards sont désormais centralisés dans l'onglet `sys\_ID\_Fichiers` (valeur par défaut), tandis que les colonnes dans `Paramètres Généraux` ne servent qu'à définir des exceptions.

5. \*\*Phase de Débogage Intense\*\* :

\* \*\*Détection de Langue\*\* : Nous avons corrigé un bug dans la fonction `getOriginalLanguage` qui ne détectait pas correctement l'anglais, ce qui empêchait l'envoi des e-mails de résultats.

\* \*\*Interface de Retraitement\*\* : Nous avons diagnostiqué et corrigé plusieurs problèmes "silencieux" qui empêchaient l'affichage de la barre latérale de retraitement, en rapatriant la logique d'ouverture dans le script `Menu.gs`.

\* \*\*Préférences Utilisateur\*\* : Nous avons ajusté l'interface de retraitement pour revenir à une saisie manuelle systématique du numéro de ligne, selon votre préférence.

\* \*\*Logique de Retraitement\*\* : Nous avons affiné le script pour qu'il n'envoie plus l'e-mail de confirmation lors d'un retraitement manuel.

\* \*\*Destinataires Multiples\*\* : Nous avons corrigé le bug final où les destinataires supplémentaires (formateur, patron) cochés dans l'interface de retraitement n'étaient pas pris en compte.

### 2. État Actuel du Projet

\* \*\*Tâches réalisées\*\* :

\* Développement complet, débogage et fiabilisation de la fonctionnalité d'envoi d'e-mail de confirmation multilingue et personnalisable.

\* Mise en place d'une architecture de configuration robuste (valeur par défaut + écrasement) pour les gabarits d'e-mail.

\* Correction de plusieurs bugs liés à la détection de la langue et à l'interface de retraitement.

\* \*\*Tâches en cours\*\* :

\* Aucune. La fonctionnalité "E-mail de confirmation" est considérée comme terminée.

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

\* \*\*Validation finale (par vous)\*\* : La toute première action de notre prochaine session sera de valider que la dernière version du script (`TraitementReponses.gs` v18.5) envoie bien l'e-mail de résultats au formateur/patron lors d'un retraitement.

\* \*\*Développement "Envoi Différé"\*\* : Entamer le développement de la deuxième brique des "Évolutions E-mail" : la possibilité de programmer l'envoi des résultats après un certain délai.

### 3. Contexte Historique du Projet

Partant d'un moteur de traitement dynamique et stable (Version 16.0), cette session a marqué le début de la phase d'amélioration de la communication et de l'expérience utilisateur. Nous avons mis en pause l'intégration de nouveaux contenus (test "R&K") pour nous concentrer sur l'ajout de fonctionnalités demandées par les cas d'usage métier, comme la confirmation immédiate de la prise en compte des réponses. Cette session a parfaitement illustré le passage de la construction du "moteur" à la construction de la "carrosserie" et des fonctionnalités de confort de l'usine.

### 4. Orientations Stratégiques

La session a renforcé un principe clé : \*\*la robustesse par la centralisation\*\*. La décision de déplacer les ID des gabarits par défaut dans l'onglet `sys\_ID\_Fichiers` est une application directe de la philosophie "Zéro Convention Implicite". Le comportement standard du système n'est plus implicite ou dépendant d'un copier-coller manuel, il est défini explicitement à un seul endroit. Les `Paramètres Généraux` ne servent plus qu'à déclarer les exceptions, rendant la configuration à la fois plus simple et plus claire.

### 5. Architecture Technique & Ressources Clés

#### 5.1. État des Scripts et Fichiers

\*\*Fichier 1 : `TraitementReponses.gs` (Version 18.5 - Finale)\*\*

```javascript

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

// == FICHIER : TraitementReponses.gs

// == VERSION : 18.5 (Correction de la gestion des destinataires en mode retraitement)

// == RÔLE : Gère la logique de traitement des réponses et aiguille vers le bon moteur.

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

/\*\*

\* Nettoie une chaîne de caractères pour la rendre utilisable comme nom de variable/placeholder.

\*/

function \_nettoyerEnTete(enTete) {

if (!enTete) return "";

const accents = 'ÀÁÂÃÄÅàáâãäåÒÓÔÕÖØòóôõöøÈÉÊËèéêëÇçÌÍÎÏìíîïÙÚÛÜùúûüÿÑñ';

const sansAccents = 'AAAAAAaaaaaaOOOOOOooooooEEEEeeeeCcIIIIiiiiUUUUuuuuyNn';

return enTete.toString().split('').map((char) => {

const accentIndex = accents.indexOf(char);

return accentIndex !== -1 ? sansAccents[accentIndex] : char;

}).join('')

.replace(/[^a-zA-Z0-9\_]/g, '\_');

}

/\*\*

\* Crée un objet de réponse standardisé à partir d'un numéro de ligne.

\*/

function \_creerObjetReponse(rowIndex) {

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) => {

let cle = header;

if (header && !header.includes(':')) {

cle = \_nettoyerEnTete(header);

}

if (cle) {

reponse[cle] = rowValues[i];

}

});

return reponse;

}

/\*\*

\* 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}`);

}

}

/\*\*

\* Envoie un e-mail de confirmation.

\*/

function \_envoyerEmailDeConfirmation(config, reponse, langueCible) {

try {

const nomColonneOverride = `ID\_Gabarit\_Email\_Confirmation\_${langueCible}`;

let idGabaritConfirmation = config[nomColonneOverride];

if (!idGabaritConfirmation || String(idGabaritConfirmation).trim() === '') {

const systemIds = getSystemIds();

const nomCleDefaut = `ID\_GABARIT\_CONFIRMATION\_${langueCible}`;

idGabaritConfirmation = systemIds[nomCleDefaut];

Logger.log(`Utilisation du gabarit de confirmation PAR DÉFAUT pour la langue ${langueCible}.`);

} else {

Logger.log(`Utilisation du gabarit de confirmation SPÉCIFIQUE pour la langue ${langueCible}.`);

}

const emailRepondant = reponse.Votre\_adresse\_e\_mail || reponse.Adresse\_e\_mail || reponse.emailRepondant;

if (!idGabaritConfirmation || String(idGabaritConfirmation).trim() === '' || !emailRepondant) {

Logger.log(`Aucun e-mail de confirmation à envoyer pour la langue ${langueCible}.`);

return;

}

const doc = DocumentApp.openById(idGabaritConfirmation);

let sujet = doc.getName();

const url = "[https://docs.google.com/feeds/download/documents/export/Export?id=](https://docs.google.com/feeds/download/documents/export/Export?id=)" + idGabaritConfirmation + "&exportFormat=html";

const token = ScriptApp.getOAuthToken();

const response = UrlFetchApp.fetch(url, { headers: { 'Authorization': 'Bearer ' + token } });

let corpsHtml = response.getContentText();

for (const key in reponse) {

if (reponse.hasOwnProperty(key)) {

const placeholder = `{{${key}}}`;

const valeur = reponse[key] || '';

const regex = new RegExp(placeholder.replace(/[-\/\\^$\*+?.()|[\]{}]/g, '\\$&'), 'g');

sujet = sujet.replace(regex, valeur);

corpsHtml = corpsHtml.replace(regex, valeur);

}

}

const mailOptions = { to: emailRepondant, subject: sujet, htmlBody: corpsHtml };

if (config.Email\_Alias && config.Email\_Alias.trim() !== '') {

mailOptions.from = config.Email\_Alias;

}

GmailApp.sendEmail(mailOptions.to, mailOptions.subject, "", mailOptions);

Logger.log(`E-mail de confirmation [${langueCible}] envoyé avec succès à ${emailRepondant}.`);

} catch (e) {

Logger.log(`ERREUR lors de l'envoi de l'e-mail de confirmation : ${e.toString()}\n${e.stack}`);

}

}

/\*\*

\* COEUR LOGIQUE : Aiguille le traitement vers le moteur.

\*/

function traiterLigne(rowIndex, optionsSurcharge = {}) {

try {

const config = getTestConfiguration();

const reponse = \_creerObjetReponse(rowIndex);

const langueOrigine = getOriginalLanguage(reponse);

const langueCible = optionsSurcharge.langue || langueOrigine;

if (!optionsSurcharge.isRetraitement) {

\_envoyerEmailDeConfirmation(config, reponse, langueCible);

}

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

if (config.Moteur\_Calcul === 'Universel') {

Logger.log("Moteur UNIVERSEL détecté pour la ligne " + rowIndex);

assemblerEtEnvoyerEmailUniversel(config, reponse, resultats, langueCible, optionsSurcharge);

} else {

// Moteur legacy...

}

} catch (err) {

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

}

}

/\*\*

\* Lit la BDD, assemble et envoie l'e-mail de résultats.

\*/

function assemblerEtEnvoyerEmailUniversel(config, reponse, resultats, langueCible, optionsSurcharge = {}) {

const typeTest = config.Type\_Test;

let codeNiveauEmail = config.ID\_Gabarit\_Email\_Repondant.replace('RESULTATS\_', '');

if (optionsSurcharge && optionsSurcharge.niveau && optionsSurcharge.niveau !== '') {

codeNiveauEmail = optionsSurcharge.niveau;

}

const profilFinal = resultats.profilFinal;

const systemIds = getSystemIds();

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

const compoSheet = bdd.getSheetByName("sys\_Composition\_Emails");

const compoData = compoSheet.getDataRange().getValues();

const compoHeaders = compoData.shift();

const idx = {

typeTest: compoHeaders.indexOf('Type\_Test'),

langue: compoHeaders.indexOf('Code\_Langue'),

niveau: compoHeaders.indexOf('Code\_Niveau\_Email'),

profil: compoHeaders.indexOf('Code\_Profil'),

element: compoHeaders.indexOf('Element'),

ordre: compoHeaders.indexOf('Ordre'),

contenu: compoHeaders.indexOf('Contenu / ID\_Document')

};

let briquesDeContenu = compoData.filter(row => {

const typeLigne = (row[idx.typeTest] || '').trim();

const typeCible = (typeTest || '').trim();

const typeMatch = (typeLigne === typeCible || typeLigne === '');

const langMatch = row[idx.langue] === langueCible;

const levelMatch = row[idx.niveau].includes(codeNiveauEmail);

const profilLigne = (row[idx.profil] || '').trim();

const profilCible = (profilFinal || '').trim();

const profileMatch = (profilLigne === profilCible || profilLigne === '');

return typeMatch && langMatch && levelMatch && profileMatch;

});

briquesDeContenu.sort((a, b) => a[idx.ordre] - b[idx.ordre]);

let contenuInfoCopie = null;

const indexInfoCopie = briquesDeContenu.findIndex(brique => (brique[idx.element] || '').trim() === 'Info\_Copie');

if (indexInfoCopie > -1) {

contenuInfoCopie = briquesDeContenu[indexInfoCopie][idx.contenu];

briquesDeContenu.splice(indexInfoCopie, 1);

}

let sujet = `Résultats de votre test ${typeTest}`;

let corpsHtml = "";

const piecesJointesIds = new Set();

for (const brique of briquesDeContenu) {

const elementType = (brique[idx.element] || '').trim();

const contenu = brique[idx.contenu];

switch (elementType) {

case 'Sujet\_Email': sujet = contenu; break;

case 'Introduction': case 'Corps\_Texte': corpsHtml += contenu + "<br>"; break;

case 'Document': if (contenu) piecesJointesIds.add(contenu); break;

case 'Ligne\_Score':

Object.entries(resultats.scoresData).sort((a, b) => b[1] - a[1]).forEach(([code, score]) => {

let ligneScore = contenu.replace(/{{nom\_profil}}/g, resultats.mapCodeToName[code] || code).replace(/{{score}}/g, score);

corpsHtml += ligneScore + "<br>";

});

break;

}

}

const donneesPourEmail = { ...reponse, ...resultats };

for (const key in donneesPourEmail) {

if (donneesPourEmail.hasOwnProperty(key)) {

const placeholder = `{{${key}}}`;

const valeur = donneesPourEmail[key] || '';

const regex = new RegExp(placeholder.replace(/[-\/\\^$\*+?.()|[\]{}]/g, '\\$&'), 'g');

sujet = sujet.replace(regex, valeur);

corpsHtml = corpsHtml.replace(regex, valeur);

if (contenuInfoCopie) {

contenuInfoCopie = contenuInfoCopie.replace(regex, valeur);

}

}

}

const piecesJointes = Array.from(piecesJointesIds).map(id => {

try { return DriveApp.getFileById(id).getBlob(); }

catch(e) { Logger.log(`ERREUR PJ : ${id}. Détails : ${e.message}`); return null; }

}).filter(Boolean);

const T = loadTraductions(langueCible);

const adressesUniques = new Set();

const emailRepondantPrincipal = reponse.Votre\_adresse\_e\_mail || reponse.Adresse\_e\_mail || reponse.emailRepondant;

const destinatairesSurcharge = optionsSurcharge.destinataires || {};

const useSurcharge = Object.keys(destinatairesSurcharge).length > 0;

if (useSurcharge) {

if (destinatairesSurcharge.repondant && emailRepondantPrincipal) { adressesUniques.add(emailRepondantPrincipal); }

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 (config.Repondant\_Email\_Actif === 'Oui' && emailRepondantPrincipal) { adressesUniques.add(emailRepondantPrincipal); }

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;

let corpsHtmlFinal = corpsHtml;

if (adresse.toLowerCase() !== (emailRepondantPrincipal || "").toLowerCase()) {

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

if (contenuInfoCopie) { corpsHtmlFinal = contenuInfoCopie + corpsHtml; }

}

const mailOptions = { to: adresse, subject: sujetFinal, htmlBody: corpsHtmlFinal, attachments: piecesJointes };

const aliasExpediteur = optionsSurcharge.alias || config.Email\_Alias;

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

mailOptions.from = aliasExpediteur;

}

GmailApp.sendEmail(mailOptions.to, mailOptions.subject, "", mailOptions);

Logger.log(`E-mail de RÉSULTATS [${langueCible}] envoyé avec succès à ${adresse}.`);

} catch (e) {

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

}

});

}

/\*\*

\* Fournit les données nécessaires à la barre latérale de retraitement.

\*/

function getDonneesPourRetraitement(rowIndex) {

try {

const config = getTestConfiguration();

const reponse = \_creerObjetReponse(rowIndex);

const nomRepondant = reponse.Votre\_nom\_et\_prenom || reponse.Nom\_et\_prenom || '';

const emailRepondant = reponse.Votre\_adresse\_e\_mail || reponse.Adresse\_e\_mail || '';

return {

nomRepondant: nomRepondant,

emailRepondant: emailRepondant,

langueOrigine: getOriginalLanguage(reponse),

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

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

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

emailAlias: config.Email\_Alias || ''

};

} catch (e) {

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

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

}

}

/\*\*

\* Lance le retraitement depuis l'interface utilisateur.

\*/

function lancerRetraitementDepuisUI(options) {

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

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

}

options.isRetraitement = true;

traiterLigne(options.rowIndex, options);

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

}

**Fichier 2 : Menu.gs (Version 2.3 - Finale)**

JavaScript

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

// == FICHIER : Menu.gs

// == VERSION : 2.3 (Retour à la saisie manuelle systématique du N° de ligne)

// == RÔLE : Crée le menu et gère l'ouverture de l'interface de retraitement.

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

/\*\*

\* S'exécute à l'ouverture de la feuille de calcul pour créer le menu personnalisé.

\*/

function onOpen() {

SpreadsheetApp.getUi()

.createMenu('⚙️ Actions Usine')

.addItem("Activer le traitement auto", "activerTraitementAutomatique")

.addSeparator()

.addItem("Retraiter une réponse...", "retraiterReponse\_UI")

.addToUi();

}

/\*\*

\* Ouvre une boîte de dialogue pour demander le numéro de la ligne à retraiter.

\*/

function retraiterReponse\_UI() {

const ui = SpreadsheetApp.getUi();

const htmlOutput = HtmlService.createHtmlOutputFromFile('DialogueLigne.html')

.setWidth(350)

.setHeight(160);

ui.showModalDialog(htmlOutput, 'Retraitement de Réponse');

}

/\*\*

\* Ouvre la barre latérale de retraitement pour une ligne donnée.

\*/

function ouvrirSidebarPourLigne(rowIndex) {

const ui = SpreadsheetApp.getUi();

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

template.ligneActive = rowIndex;

const htmlOutput = template.evaluate()

.setTitle("Retraitement - Ligne " + rowIndex)

.setWidth(350);

ui.showSidebar(htmlOutput);

}

/\*\*

\* Crée le déclencheur "onFormSubmit" pour le traitement automatique.

\*/

function activerTraitementAutomatique() {

const ss = SpreadsheetApp.getActive();

const triggers = ScriptApp.getUserTriggers(ss);

triggers.forEach(trigger => {

if (trigger.getHandlerFunction() === 'onFormSubmit') {

ScriptApp.deleteTrigger(trigger);

}

});

ScriptApp.newTrigger('onFormSubmit')

.forSpreadsheet(ss)

.onFormSubmit()

.create();

SpreadsheetApp.getUi().alert('✅ Déclencheur activé ! Le traitement automatique des réponses est maintenant opérationnel.');

}

**Fichier 3 : Utilities.gs (Version 8.1 - Finale)**

JavaScript

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

// == FICHIER : Utilities.gs - VERSION 8.1 (Correction de la détection de langue)

// == RÔLE : Boîte à outils du Kit de Traitement.

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

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);

}

}

/\*\*

\* Détecte correctement la langue de la réponse initiale de l'utilisateur.

\*/

function getOriginalLanguage(reponses) {

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

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

return mapLangue[langueRepondantBrute] || 'FR';

}

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' || !resultats.scoresData || Object.keys(resultats.scoresData).length === 0) {

return "";

}

try {

const systemIds = getSystemIds();

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

const formatSheet = bdd.getSheetByName("sys\_Formatage\_Scores");

if (!formatSheet) return "Erreur: Onglet 'sys\_Formatage\_Scores' introuvable.\n";

const formatData = formatSheet.getDataRange().getValues();

const formatHeaders = formatData.shift();

const typeTestCol = formatHeaders.indexOf('Type\_Test');

const regle = formatData.find(row => row[typeTestCol] === typeTest);

if (!regle) return `Aucune règle d'affichage trouvée pour le test '${typeTest}'.\n`;

const regleMap = {};

formatHeaders.forEach((h, i) => regleMap[h] = regle[i]);

const T = loadTraductions(langueCode);

let scoresText = (regleMap.Texte\_Intro || "Voici le détail de vos scores :") + "\n";

if (regleMap.Mode\_Affichage === 'Simple') {

let scoresArray = Object.entries(resultats.scoresData).map(([code, score]) => ({

code\_profil: code,

nom\_profil: resultats.mapCodeToName[code] || code,

score: score

}));

if (regleMap.Tri\_Scores === 'Décroissant') {

scoresArray.sort((a, b) => b.score - a.score);

} else if (regleMap.Tri\_Scores === 'Croissant') {

scoresArray.sort((a, b) => a.score - b.score);

}

scoresArray.forEach(item => {

let ligne = regleMap.Format\_Ligne.replace(/{{nom\_profil}}/g, item.nom\_profil)

.replace(/{{score}}/g, item.score)

.replace(/{{suffixe\_points}}/g, T.SUFFIXE\_POINTS || 'points');

scoresText += ligne + "\n";

});

} else if (regleMap.Mode\_Affichage === 'Dichotomie') {

const axes = [

{ nom: (T.AXE\_EI || "Extraversion (E) vs Introversion (I)"), p1: 'E', p2: 'I' },

{ nom: (T.AXE\_SN || "Sensation (S) vs Intuition (N)"), p1: 'S', p2: 'N' },

{ nom: (T.AXE\_TF || "Pensée (T) vs Sentiment (F)"), p1: 'T', p2: 'F' },

{ nom: (T.AXE\_JP || "Jugement (J) vs Perception (P)"), p1: 'J', p2: 'P' }

];

axes.forEach(axe => {

let ligne = regleMap.Format\_Ligne.replace(/{{axe\_nom}}/g, axe.nom)

.replace(/{{score1}}/g, resultats.scoresData[axe.p1] || 0)

.replace(/{{score2}}/g, resultats.scoresData[axe.p2] || 0);

scoresText += ligne + "\n";

});

}

return scoresText;

} catch (e) {

Logger.log(`ERREUR CRITIQUE DANS formatScoresDetails (universel): ${e.toString()}`);

return "Impossible d'afficher le détail des scores en raison d'une erreur.\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 || '',

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, valeur);

}

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);

}

}

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(String(niveauPJ).replace(/[^0-9]/g, ''), 10) || 1;

const idsFichiersTrouves = new Set();

data.forEach(row => {

const typeSheet = row[idx.type] ? row[idx.type].toString().toUpperCase() : '';

const typeTestUpper = typeTest ? typeTest.toUpperCase() : '';

const typeMatch = (typeSheet === typeTestUpper);

const profilSheet = row[idx.profil] ? row[idx.profil].toString().toUpperCase() : '';

const profilCodeUpper = profilCode ? profilCode.toUpperCase() : '';

const profilMatch = (profilSheet === profilCodeUpper || profilSheet === 'TOUS');

const langueSheet = row[idx.langue] ? row[idx.langue].toString().toUpperCase() : '';

const langueCodeUpper = langueCode ? langueCode.toUpperCase() : '';

const langueMatch = (langueSheet === langueCodeUpper || langueSheet === 'TOUS');

const niveauMatch = (row[idx.niveau] > 0 && row[idx.niveau] <= niveauNumRequis);

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;

}

**5.2. Architecture Détaillée et Traçabilité**

* **MODIFIÉ** : TraitementReponses.gs (v18.5) -> Ajout d'une fonction de lecture centralisée (\_creerObjetReponse), modification de la logique d'envoi de confirmation et correction de la gestion des destinataires en retraitement.
* **MODIFIÉ** : Menu.gs (v2.3) -> Simplification du menu pour revenir à la saisie manuelle systématique du numéro de ligne.
* **MODIFIÉ** : Utilities.gs (v8.1) -> Correction de la logique de détection de la langue dans getOriginalLanguage.
* **AJOUTÉ** ([CONFIG]V2 Usine à Tests > sys\_ID\_Fichiers) :
  + ID\_GABARIT\_CONFIRMATION\_FR
  + ID\_GABARIT\_CONFIRMATION\_EN
  + ID\_GABARIT\_CONFIRMATION\_ES
  + ...etc.
* **AJOUTÉ** ([CONFIG]V2 Usine à Tests > Paramètres Généraux) :
  + ID\_Gabarit\_Email\_Confirmation\_FR
  + ID\_Gabarit\_Email\_Confirmation\_EN
  + ID\_Gabarit\_Email\_Confirmation\_ES
  + ...etc.

**5.3. Nomenclature des Variables**

* **Nom Précis** : ID\_GABARIT\_CONFIRMATION\_[CODE\_LANGUE]
* **Type** : String
* **Origine/Portée** : Clé dans l'onglet sys\_ID\_Fichiers. Contient l'ID du Google Doc servant de gabarit de confirmation par défaut pour une langue donnée.
* **Contenu Attendu** : Un ID de Google Doc valide.
* **Exemple Concret** : "1DALk42jmUhSZx9XqjhCjxoVjMxYkDky8kfTSXxDHmXk"

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

* **Fichier** : [CONFIG]V2 Usine à Tests
* **Onglet** : sys\_ID\_Fichiers
* **Colonnes Critiques** : Clé (doit contenir ID\_GABARIT\_CONFIRMATION\_FR, etc.), ID.
* **Fichier** : [CONFIG]V2 Usine à Tests
* **Onglet** : Paramètres Généraux
* **Colonnes Critiques** : ID\_Gabarit\_Email\_Confirmation\_FR, etc. (utilisées pour surcharger le gabarit par défaut).

**6. Actions Recommandées / Prochaines Étapes**

* **Action Immédiate (Pour Vous)** :
  + Effectuer un dernier test de la fonction "Retraiter une réponse..." en cochant la case "Envoyer au Formateur" (ou autre) et en renseignant une adresse e-mail.
  + **Vérifier la réception de l'e-mail par le formateur**, car il s'agit du dernier bug que nous avons identifié et corrigé.
* **Questions pour notre prochaine session** :
  + La fonctionnalité de retraitement est-elle maintenant 100% fonctionnelle et conforme à vos attentes ?
  + Pouvons-nous considérer la brique "E-mail de confirmation" comme terminée et validée ?
  + Sommes-nous prêts à démarrer le développement de la brique "Envoi Différé des Résultats" ?