**De Projet: Usine à Tests Date: 2025-07-14 (session 2) Version: 4.0**

**1. Résumé des Échanges du Jour**

La session d'aujourd'hui a été consacrée au premier test de bout en bout de la chaîne V2, ce qui a permis de révéler et de corriger plusieurs incohérences critiques entre les différents composants du système.

Le processus de débogage a suivi plusieurs étapes clés :

1. Erreur de Génération du Formulaire : Le premier test a échoué car le [MOTEUR] lisait l'ancienne colonne TypeQuestion au lieu de la clé "mode" du JSON. Le script Moteur V2.js a été corrigé pour se conformer au principe "JSON comme source de vérité".
2. Gestion des Questions Standards (Blocs Méta) : Nous avons constaté que la question "Email" n'était pas ajoutée. Cela a initié une discussion approfondie sur la fonctionnalité des "Blocs Méta", menant à plusieurs itérations :
   * Solution 1 (Manuelle) : Ajout manuel de la colonne Blocs\_Meta\_A\_Inclure dans [CONFIG] Usine à Tests.
   * Solution 2 (Interface Statique) : Modification du formulaire de configuration pour inclure un champ de texte pré-rempli avec META\_EMAIL.
   * Solution 3 (Finale et Dynamique) : Sur votre proposition, nous avons opté pour une solution bien plus élégante. L'interface de configuration lit désormais le catalogue de questions disponibles dans l'onglet Questions\_META\_FR de la [BDD] et les présente sous forme de cases à cocher, offrant une flexibilité et une simplicité maximales.
3. Correction du Calcul des Scores : Un test a révélé que les calculs ne se faisaient pas. La cause était une incohérence majeure : le [MOTEUR] lisait le JSON pour créer le formulaire, mais le [TEMPLATE] (Logique\_Universel.gs) lisait encore l'ancienne colonne TypeQuestion pour calculer les scores. Le script a été corrigé pour se baser uniquement sur le "mode" du JSON.
4. Fiabilisation de la Restitution : L'e-mail de résultat affichait "Bonjour null,". Le script TraitementReponses.gs a été rendu plus robuste en cherchant les colonnes "Nom" et "Email" par mots-clés plutôt que par des titres exacts.
5. Clarification de l'Architecture : Nous avons affiné la définition de la "logique hybride" du [MOTEUR], qui traite META\_EMAIL et META\_NOM\_COMPLET comme des instructions spéciales intégrées, tout en conservant la capacité de lire des blocs personnalisés depuis la [BDD].
6. Catalogue de Blocs Méta : Pour finaliser, un catalogue de questions standards (Consentement, Email Manager, etc.) a été proposé pour peupler l'onglet Questions\_META\_FR et exploiter pleinement la nouvelle interface de configuration dynamique.

**2. État Actuel du Projet**

* Tâches réalisées :
  + Correction de la logique de lecture du type de question dans [MOTEUR] Moteur V2.js.
  + Correction de la logique de lecture du type de question dans [TEMPLATE] Logique\_Universel.gs, alignant ainsi le calcul sur la génération.
  + Correction de la logique de récupération de l'e-mail et du nom dans [TEMPLATE] TraitementReponses.gs.
  + Mise à jour majeure de l'interface de configuration ([CONFIG] menu.gs et FormulaireUI.html) pour proposer une sélection dynamique des blocs méta par cases à cocher.
  + Définition d'un catalogue de questions standards pour l'onglet Questions\_META\_FR.
* Tâches en cours :
  + Génération de ce document de synthèse.
* Prochaines étapes planifiées :
  + Test final de bout en bout : Lancer une dernière fois la chaîne complète en utilisant la nouvelle interface de configuration dynamique pour valider l'ensemble des correctifs.

**3. Contexte Historique du Projet**

S'appuyant sur la migration des données effectuée précédemment, la session d'aujourd'hui a permis de passer de la "compatibilité théorique" à la "robustesse fonctionnelle". Les tests ont révélé des incohérences logiques entre les différents composants ([MOTEUR], [CONFIG], [TEMPLATE]), qui ont toutes été résolues. Le projet a atteint une maturité où la chaîne de production est non seulement fonctionnelle, mais aussi intuitive et flexible pour l'utilisateur final.

**4. Orientations Stratégiques**

La session a été une mise en application parfaite des principes directeurs de la V2 :

* JSON comme Source Unique de Vérité : Le débogage a imposé une discipline stricte, forçant le [MOTEUR] et le [TEMPLATE] à lire la même source de vérité ("mode" dans le JSON), éliminant ainsi une incohérence majeure.
* Zéro Convention Implicite : La transformation du champ Blocs\_Meta\_A\_Inclure en une interface à cases à cocher est l'aboutissement de ce principe. L'utilisateur déclare maintenant explicitement chaque question standard qu'il souhaite inclure, de manière claire et sans ambiguïté.
* Le Code plus Intelligent que la Configuration : La logique hybride du [MOTEUR] et la recherche par mot-clé dans le [TEMPLATE] sont des exemples de code conçu pour être robuste face à des configurations variées.

**5. Architecture Technique & Ressources Clés**

**5.1. État des Scripts et Fichiers**

Voici les versions finales et complètes des scripts modifiés aujourd'hui.

Projet [MOTEUR] Usine à Tests V2

Fichier Moteur V2.js (MODIFIÉ)

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

// FICHIER : Moteur V2.js

// RÔLE : Fonctions principales de création et d'orchestration des tests.

// VERSION : 4.4 - Logique hybride pour les Blocs Méta (spécial/BDD)

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

function lancerCreationFormulaire() { /\* ... (inchangé) ... \*/ }

function lancerCreationSysteme(rowIndex) {

  try {

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

    const config = getConfigurationFromRow(rowIndex);

    if (config['Statut'].toLowerCase() !== 'en construction') { return null; }

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

    const systemIds = getSystemIds();

    if (!systemIds.ID\_TEMPLATE\_TRAITEMENT\_V2) { throw new Error("Clé 'ID\_TEMPLATE\_TRAITEMENT\_V2' introuvable."); }

    let dossierCible;

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

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

    } else {

      if (!systemIds.ID\_DOSSIER\_CIBLE\_GEN) { throw new Error("Clé 'ID\_DOSSIER\_CIBLE\_GEN' introuvable."); }

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

    }

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

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

    const form = FormApp.create(nomFichierComplet);

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

    form.setProgressBar(true);

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

    DriveApp.getFileById(form.getId()).moveTo(dossierCible);

    try {

      if (!systemIds.ID\_BDD) { throw new Error("Clé 'ID\_BDD' introuvable."); }

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

// --- DÉBUT MODIFICATION V4.4 ---

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

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

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

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

let metaQuestionsMap = {};

if (metaSheet) {

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

const metaHeaders = metaData.shift();

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

if (idCol !== -1) {

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

acc[row[idCol].toUpperCase()] = row;

return acc;

}, {});

}

}

        metaIds.forEach(id => {

switch(id) {

case 'META\_EMAIL':

creerItemFormulaire(form, 'TEXTE\_EMAIL', 'Adresse Email', null, null, null);

break;

case 'META\_NOM\_COMPLET':

creerItemFormulaire(form, 'TEXTE\_COURT', 'Nom Complet', null, null, null);

break;

default:

if (metaQuestionsMap[id]) {

const q\_data = metaQuestionsMap[id];

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

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

if (q\_params\_json) {

try {

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

if (parsedJson.mode) { final\_meta\_type = parsedJson.mode; }

} catch (e) { console.warn(`Bloc Méta BDD '${id}': JSON invalide.`); }

}

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

} else {

console.warn(`Bloc méta ID '${id}' non trouvé.`);

}

break;

}

        });

      }

// --- FIN MODIFICATION V4.4 ---

      const toutesLesFeuillesBDD = bdd.getSheets();

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

      const languesAInclure = [];

      toutesLesFeuillesBDD.forEach(feuille => {

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

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

      });

      if (languesAInclure.length === 0) { throw new Error("Aucune feuille de questions trouvée pour le type '" + config['Type\_Test'] + "'."); }

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

      const choices = [];

      languesAInclure.forEach(langue => {

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

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

        let nbQuestionsAUtiliser = langue.feuille.getLastRow() - 1;

        if (config['nbQuestions'] && config['nbQuestions'] > 0) { nbQuestionsAUtiliser = Math.min(config['nbQuestions'], nbQuestionsAUtiliser); }

        if (nbQuestionsAUtiliser <= 0) return;

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

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

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

            let final\_type = type\_old;

            if (params\_json) {

                try {

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

                    if (parsedJson.mode) { final\_type = parsedJson.mode; }

                } catch (e) { console.warn(`Ligne ${index + 2}: JSON invalide.`); }

            }

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

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

        });

      });

      itemLangue.setChoices(choices);

    } catch (e) { throw new Error("Génération des questions impossible. Erreur : " + e.message); }

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

  } catch(e) {

    SpreadsheetApp.getUi().alert("Erreur ligne " + rowIndex + ": " + e.message);

    throw e;

  }

}

function orchestrateurCreationAutomatique() { /\* ... (inchangé) ... \*/ }

function lancerDeploiementComplet(rowIndex) { /\* ... (inchangé) ... \*/ }

Fichier Utils V2.gs (MODIFIÉ)

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

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

// VERSION : 2.5 - Ajout de la gestion des types TEXTE\_EMAIL et TEXTE\_COURT

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

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

function getSystemIds() { /\* ... (inchangé) ... \*/ }

function getConfigurationFromRow(rowIndex) { /\* ... (inchangé) ... \*/ }

function getLangueFullName(code) { /\* ... (inchangé) ... \*/ }

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

    let params = null;

    let choices = [];

    if (paramsJSONString) {

        try { params = JSON.parse(paramsJSONString); }

catch (e) { form.addParagraphTextItem().setTitle("[Erreur JSON] " + titre); return; }

    }

    if (params && params.options) {

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

    } else if (optionsString) {

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

    }

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

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

        if (choices.length > 0) { form.addCheckboxItem().setTitle(titre).setChoiceValues(choices).setRequired(true); }

else { form.addParagraphTextItem().setTitle("[Erreur QRM] " + titre); }

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

        if (choices.length > 0) { form.addMultipleChoiceItem().setTitle(titre).setChoiceValues(choices).setRequired(true); }

else { form.addParagraphTextItem().setTitle("[Erreur QCU] " + titre); }

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

/\* ... (inchangé) ... \*/

// --- DÉBUT MODIFICATION V2.5 ---

    } 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();

textItem.setValidation(emailValidation);

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

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

// --- FIN MODIFICATION V2.5 ---

    } else {

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

    }

}

Projet [CONFIG] Usine à Tests

Fichier menu.gs (MODIFIÉ)

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

// FICHIER : menu.gs (Version Harmonisée)

// VERSION : 3.1 - Logique dynamique pour Blocs Méta

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

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

function onOpen() { /\* ... (inchangé) ... \*/ }

function showConfigurationSidebar() { /\* ... (inchangé) ... \*/ }

function getInitialData() {

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

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

  if (!optionsSheet) { throw new Error("Onglet 'sys\_Options\_Parametres' introuvable."); }

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

  const headers = optionsData.shift();

  const optionsMap = {};

  headers.forEach((header, i) => {

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

    optionsMap[header] = options;

  });

let availableMetaBlocks = [];

try {

const systemIds = getSystemIds();

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

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

if (metaSheet) {

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

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

}

} catch(e) { console.error("Impossible de charger les blocs méta : " + e.message); }

  return {

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

availableMetaBlocks: availableMetaBlocks,

    options: { /\* ... (inchangé) ... \*/ }

  };

}

function getQuestionCountForTestType(typeTest) { /\* ... (inchangé) ... \*/ }

function processNewTestConfiguration(formObject) {

  try {

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

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

const newHeaderName = 'Blocs\_Meta\_A\_Inclure';

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

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

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

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

}

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

    const dataRow = {

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

      'Statut': 'En construction',

      'Type\_Test': formObject.type,

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

      'nbQuestions': formObject.nbQuestions,

/\* ... (autres champs inchangés) ... \*/

    };

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

    paramsSheet.appendRow(nouvelleLigne);

    return "Configuration enregistrée avec succès !";

  } catch (e) { throw new Error("Sauvegarde impossible. " + e.message); }

}

function getSystemIds() { /\* ... (inchangé) ... \*/ }

Fichier FormulaireUI.html (MODIFIÉ)

<!-- ================================================================================= -->

<!-- FICHIER : FormulaireUI.html -->

<!-- VERSION : 3.1 - Correction de l'affichage du nombre de questions -->

<!-- ================================================================================= -->

<!DOCTYPE html>

<html>

  <head>

    <base target="\_top">

    <script src="[https://cdn.tailwindcss.com](https://cdn.tailwindcss.com)"></script>

    <style>

      .hidden-section { max-height: 0; overflow: hidden; transition: all 0.5s ease-out; opacity: 0; }

      .visible-section { max-height: 500px; opacity: 1; }

    </style>

  </head>

  <body class="bg-gray-100 p-4 font-sans">

    <div id="main-form">

<!-- Titre, Type de Test (inchangés) -->

<div class="mb-4">

<label class="block text-sm font-medium text-gray-700">Questions Standards (Blocs Méta)</label>

<div id="meta-blocks-container" class="mt-2 p-3 bg-white border rounded-md space-y-2"></div>

</div>

<div id="questions-info" class="mb-4 p-3 bg-indigo-50 border rounded-md hidden">

        <p class="text-sm text-indigo-700">Nombre de questions disponibles : <span id="question-count" class="font-bold">...</span></p>

      </div>

<!-- nbQuestions, Sections Email, Bouton (inchangés) -->

    </div>

    <div id="feedback-screen" class="hidden text-center py-10"><p id="feedback-message"></p></div>

    <script>

      document.addEventListener('DOMContentLoaded', function() {

        google.script.run.withSuccessHandler(populateInitialData).getInitialData();

setupEventListeners();

      });

      function populateInitialData(data) {

/\* ... (remplissage Type de Test inchangé) ... \*/

const metaContainer = document.getElementById('meta-blocks-container');

if (data.availableMetaBlocks && data.availableMetaBlocks.length > 0) {

metaContainer.innerHTML