Voici la synthèse de notre session, prête à être sauvegardée.

Markdown

---

Projet: "Usine à Tests"

Date: "2508111021"

Version: "7.0"

---

### 1. Résumé des Échanges du Jour

La session d'aujourd'hui a marqué un tournant majeur dans l'architecture du projet, passant de corrections de bugs spécifiques à une refonte stratégique pour une robustesse et une autonomie accrues.

Initialement, nous avons corrigé une série de bugs dans l'affichage des résultats par e-mail :

1. \*\*Profil non reconnu :\*\* Nous avons diagnostiqué que la fonction `\_determinerProfilFinal` était trop simple pour les tests à seuils (type "r&K"). Nous l'avons remplacée par une fonction "routeur" qui appelle une nouvelle logique de calcul par pourcentage (`\_determinerProfilFinalParSeuils`), résolvant le premier problème.

2. \*\*Description incorrecte :\*\* Nous avons ensuite constaté que la fonction `\_chargerProfils` lisait la mauvaise colonne et ne gérait pas la structure "un-à-plusieurs" des profils complexes. La décision a été prise de \*\*standardiser les en-têtes de colonnes\*\* de tous les onglets de profils "simples" (MBTI, Couleurs, Ancres) pour fiabiliser la lecture des données.

3. \*\*Détail des scores manquant :\*\* Le dernier bug identifié était l'absence des scores détaillés pour les tests "r&K", due à une logique codée en dur dans la fonction `formatScoresDetails`.

Face à ce dernier point, vous avez proposé une évolution stratégique majeure : \*\*rendre la fonction `formatScoresDetails` entièrement pilotée par les données\*\*, au lieu de simplement y ajouter une nouvelle condition. Cette décision constitue l'avancée la plus significative de la journée, menant à la conception d'un nouveau système de formatage universel.

### 2. État Actuel du Projet

Le projet est dans une phase de transition architecturale. Les fondations d'un système entièrement piloté par les données sont posées, mais leur implémentation nécessite des actions de configuration précises.

\* \*\*Tâches réalisées (Conception) :\*\*

\* Conception d'une fonction `\_chargerProfils` robuste, capable de gérer deux familles de structures de profils (simple et complexe).

\* Conception d'une fonction `formatScoresDetails` entièrement universelle, qui lit ses instructions depuis une nouvelle feuille de configuration.

\* Définition d'un standard de nommage pour les colonnes des onglets de profils "simples".

\* Définition de la structure du nouvel onglet de configuration `sys\_Formatage\_Scores`.

\* \*\*Tâches en cours :\*\* Aucune. Le projet est en attente de la mise à jour de la base de données.

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

1. Standardiser les en-têtes des onglets de profils simples (`Profils\_Couleurs\_FR`, `Profils\_Ancres\_FR`, `Profils\_MBTI\_FR`).

2. Créer et peupler le nouvel onglet `sys\_Formatage\_Scores` dans la base de données.

3. Restructurer les données des onglets `Profils\_r&K\_...` pour les adapter à la nouvelle logique de formatage.

4. Déployer les versions finalisées des scripts `Logique\_Universel.gs` et `Utilities.gs`.

### 3. Contexte Historique du Projet

Partant d'un système fonctionnel mais rigide, le projet évolue vers une plateforme de création de tests hautement configurable. Chaque bug rencontré (URL de formulaire, calcul de profil, affichage des résultats) n'a pas mené à un simple correctif, mais à une refonte architecturale qui élimine une classe entière de problèmes potentiels. La session actuelle est l'aboutissement de cette démarche, visant à supprimer les dernières logiques "codées en dur" au profit d'un système où toute la configuration réside dans la base de données.

### 4. Orientations Stratégiques

La session renforce de manière spectaculaire les principes directeurs du projet, notamment \*\*"Zéro Convention Implicite"\*\* et \*\*"La Base de Données est la Source Unique de Vérité"\*\*.

La décision de ne pas patcher `formatScoresDetails` mais de la rendre pilotée par les données est un choix stratégique majeur. Il valide l'objectif à long terme : créer une "usine" où l'ajout ou la modification d'un test est une opération de saisie de données, ne requérant aucune modification du code. Cette approche garantit la maintenabilité, l'évolutivité et l'autonomie de l'utilisateur final.

### 5. Architecture Technique & Ressources Clés

#### 5.1. État des Scripts et Fichiers

Voici les versions finales des scripts modifiés, prêtes à être déployées \*\*après\*\* la mise à jour de la base de données.

##### Fichier : `Logique\_Universel.gs` (Version 6.0)

```javascript

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

// == FICHIER : Logique\_Universel.gs

// == VERSION : 6.0 (Gestion multi-standard des profils)

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

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

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

  let resultats = {

    profilFinal: "",

    titreProfil: "",

    descriptionProfil: "",

    scoresData: {},

    sousTotauxParMode: {}

  };

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

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

  if (!questionsMapCible) return {};

  if (langueOrigine === langueCible) {

    \_executerCalcul(reponsesUtilisateur, questionsMapCible, resultats);

  } else {

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

    if (!questionsMapOrigine) return {};

    for (const enTeteComplet in reponsesUtilisateur) {

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

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

      const questionConfigCible = questionsMapCible[idQuestion];

      if (questionConfigCible) {

        if (questionConfigCible.parametres.mode === 'ECHELLE\_NOTE') {

          \_aiguillerCalcul(questionConfigCible.parametres.mode, reponsesUtilisateur[enTeteComplet], questionConfigCible.parametres, resultats);

        } else {

          const questionConfigOrigine = questionsMapOrigine[idQuestion];

          if (questionConfigOrigine && questionConfigOrigine.parametres.options) {

            const reponsesArray = String(reponsesUtilisateur[enTeteComplet]).split(',').map(r => r.trim());

            reponsesArray.forEach(reponseSimple => {

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

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

                const optionCible = questionConfigCible.parametres.options[optionIndex];

                \_aiguillerCalcul(questionConfigCible.parametres.mode, optionCible.libelle, questionConfigCible.parametres, resultats);

              }

            });

          }

        }

      }

    }

  }

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

    resultats.profilFinal = \_determinerProfilFinal(resultats.scoresData, config.Type\_Test, langueCible);

    if (profilsMap[resultats.profilFinal]) {

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

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

        resultats.mapCodeToName = \_creerMapCodeVersNom(profilsMap);

    }

  }

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

  return resultats;

}

function \_executerCalcul(reponses, questionsMap, resultats) {

    for (const enTeteComplet in reponses) {

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

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

        const questionConfig = questionsMap[idQuestion];

        if (questionConfig) {

            \_aiguillerCalcul(questionConfig.parametres.mode, reponses[enTeteComplet], questionConfig.parametres, resultats);

        }

    }

}

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 : ${mode}`); break;

    }

}

function \_traiterQCU\_CAT(reponseUtilisateur, parametres, resultats) {

  if (!reponseUtilisateur || !parametres || !parametres.options) return;

  const optionTrouvee = parametres.options.find(opt => opt.libelle === reponseUtilisateur);

  if (optionTrouvee && optionTrouvee.profil) {

    const profil = optionTrouvee.profil;

    const valeur = (typeof optionTrouvee.valeur === 'number') ? optionTrouvee.valeur : 1;

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

  }

}

function \_determinerProfilFinalParSeuils(scoresData, typeTest, langue) {

  try {

    const systemIds = getSystemIds();

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

    const nomFeuilleProfils = `Profils\_${typeTest}\_${langue}`;

    const sheetProfils = bdd.getSheetByName(nomFeuilleProfils);

    if (!sheetProfils) return "";

    const dataProfils = sheetProfils.getRange("A2:C" + sheetProfils.getLastRow()).getValues();

    const totalPoints = Object.values(scoresData).reduce((sum, val) => sum + val, 0);

    if (totalPoints === 0) return "";

    const profilMajoritaire = Object.keys(scoresData).reduce((a, b) => scoresData[a] > scoresData[b] ? a : b);

    const scoreMajoritaire = scoresData[profilMajoritaire];

    const pourcentage = (scoreMajoritaire / totalPoints) \* 100;

    for (const row of dataProfils) {

      const nomProfil = row[0]; const conditionSeuil = row[1];

      if (!nomProfil || !conditionSeuil) continue;

      const codeProfilSeuil = conditionSeuil.split(' ')[0];

      if (codeProfilSeuil.toUpperCase() !== profilMajoritaire.toUpperCase()) continue;

      if (conditionSeuil.includes('>=')) {

        const seuil = parseFloat(conditionSeuil.replace(/[^0-9.]/g, ''));

        if (pourcentage >= seuil) return nomProfil;

      } else if (conditionSeuil.includes('<=')) {

        const seuil = parseFloat(conditionSeuil.replace(/[^0-9.]/g, ''));

        if (pourcentage <= seuil) return nomProfil;

      } else if (conditionSeuil.includes('-')) {

        const parts = conditionSeuil.match(/(\d+)-(\d+)/);

        if (parts) {

          if (pourcentage >= parseInt(parts[1], 10) && pourcentage <= parseInt(parts[2], 10)) return nomProfil;

        }

      }

    }

    return profilMajoritaire;

  } catch (e) {

    Logger.log("Erreur dans \_determinerProfilFinalParSeuils: " + e.message);

    return "";

  }

}

function \_determinerProfilFinal(scoresData, typeTest, langue) {

  if (!scoresData || Object.keys(scoresData).length === 0) return "";

  const testsAvecSeuils = ["r&K\_Adaptabilite", "r&K\_Resilience", "r&K\_Creativite"];

  if (testsAvecSeuils.some(t => typeTest.toUpperCase() === t.toUpperCase())) {

    return \_determinerProfilFinalParSeuils(scoresData, typeTest, langue);

  }

  if (typeTest.toUpperCase() === '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 \_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 profilsMap = {};

    const testsAvecSeuils = ["r&K\_Adaptabilite", "r&K\_Resilience", "r&K\_Creativite"];

    if (testsAvecSeuils.some(t => typeTest.toUpperCase() === t.toUpperCase())) {

      const profilCol = headers.indexOf('Profil');

      const titreCol = headers.indexOf('Seuil\_Score');

      const descCol = headers.indexOf('Recommandation');

      if (profilCol === -1 || titreCol === -1 || descCol === -1) throw new Error("Colonnes manquantes ('Profil', 'Seuil\_Score', 'Recommandation') pour le test " + typeTest);

      data.forEach(row => {

        const codeProfil = row[profilCol];

        if (codeProfil && !profilsMap[codeProfil]) {

          profilsMap[codeProfil] = { titre: row[titreCol] || "", description: row[descCol] || "" };

        }

      });

    } else {

      const codeCol = headers.indexOf('Code\_Profil');

      const titreCol = headers.indexOf('Titre\_Profil');

      const descCol = headers.indexOf('Description\_Profil');

      if (codeCol === -1 || titreCol === -1 || descCol === -1) throw new Error("Colonnes standardisées manquantes ('Code\_Profil', 'Titre\_Profil', 'Description\_Profil') pour le test " + typeTest);

      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 {};

  }

}

// ... autres fonctions utilitaires non modifiées ...

function \_creerMapCodeVersNom(profilsMap) { /\* ... \*/ }

function \_chargerQuestions(typeTest, langue) { /\* ... \*/ }

function \_traiterQCU\_DIRECT(reponseUtilisateur, parametres, resultats) { /\* ... \*/ }

function \_traiterQRM\_CAT(reponseUtilisateur, parametres, resultats) { /\* ... \*/ }

function \_traiterECHELLE\_NOTE(reponseUtilisateur, parametres, resultats) { /\* ... \*/ }

**Fichier : Utilities.gs (Version 8.0)**

JavaScript

// ... fonctions getTestConfiguration, getSystemIds, etc. ...

/\*\*

\* FONCTION UNIVERSELLE : Formate le bloc de scores en lisant les règles depuis la BDD.

\* Version 3.0 - Entièrement pilotée par les données.

\*/

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 "";

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 "";

const regleMap = {};

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

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

const T = loadTraductions(langueCode);

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(/{{code\_profil}}/g, item.code\_profil)

.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.\n";

}

}

// ... autres fonctions comme buildAndSendEmails, findAttachments, etc. ...

**5.2. Architecture Détaillée et Traçabilité**

* **Logique\_Universel.gs** : **MODIFIÉ PROFONDÉMENT**. La fonction \_chargerProfils est désormais un "routeur" qui gère deux types de structures de données (simple standardisée et complexe), rendant le système flexible.
* **Utilities.gs** : **MODIFIÉ PROFONDÉMENT**. La fonction formatScoresDetails est maintenant entièrement pilotée par les données de l'onglet sys\_Formatage\_Scores, éliminant toute logique codée en dur.
* **[BDD]V2 Tests & Profils** : **MODIFICATION REQUISE**.
  + **AJOUTÉ :** Un nouvel onglet sys\_Formatage\_Scores doit être créé et peuplé.
  + **MODIFIÉ :** Les en-têtes des onglets Profils\_Couleurs\_FR, Profils\_Ancres\_FR, Profils\_MBTI\_FR (et leurs équivalents dans d'autres langues) doivent être **standardisés**.
  + **À RESTRUCTURER :** Les onglets Profils\_r&K\_... doivent être repensés pour s'adapter à la nouvelle logique de formatage.

**5.3. Nomenclature des Variables**

* **{{code\_profil}}** (String) : Placeholder pour le code du profil (ex: "R", "J", "Expertise"). *Origine* : sys\_Formatage\_Scores, colonne Format\_Ligne.
* **{{nom\_profil}}** (String) : Placeholder pour le nom complet du profil (ex: "Rouge", "Le Compétiteur Déterminé"). *Origine* : sys\_Formatage\_Scores, colonne Format\_Ligne.
* **{{score}}** (Number) : Placeholder pour le score numérique d'un profil. *Origine* : sys\_Formatage\_Scores, colonne Format\_Ligne.
* **{{axe\_nom}}** (String) : Placeholder pour le nom d'un axe de dichotomie (ex: "Extraversion (E) vs Introversion (I)"). *Origine* : sys\_Formatage\_Scores, colonne Format\_Ligne (en mode Dichotomie).
* **{{score1}}, {{score2}}** (Number) : Placeholders pour les scores d'une paire de dichotomie. *Origine* : sys\_Formatage\_Scores, colonne Format\_Ligne (en mode Dichotomie).

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

* **Fichier : [BDD]V2 Tests & Profils**
  + **Nouvel Onglet : sys\_Formatage\_Scores**
    - **Colonnes critiques :** Type\_Test, Mode\_Affichage, Texte\_Intro, Format\_Ligne, Tri\_Scores.
  + **Onglets Profils Simples (MBTI, Couleurs, Ancres)**
    - **Nouveaux en-têtes standards requis :** Code\_Profil, Titre\_Profil, Description\_Profil.
  + **Onglets Profils Complexes (r&K)**
    - **Structure actuelle :** Profil, Seuil\_Score, Destinataire, Axe, Recommandation. **Cette structure est à revoir.**

**6. Actions Recommandées / Prochaines Étapes**

1. **Action Immédiate :** Standardiser les en-têtes de colonnes des onglets de profils simples (Couleurs, Ancres, MBTI) comme décrit en section 5.4.
2. **Action Suivante :** Créer l'onglet sys\_Formatage\_Scores et le remplir avec les règles pour chaque type de test.
3. **Question pour notre prochaine session :** La structure actuelle de vos onglets de profil "r&K" est très riche, avec des recommandations différentes par Destinataire (Répondant, Manager...) et par Axe (Développer potentiel, Sous stress...). Notre nouveau formateur universel ne peut pour l'instant afficher qu'**une seule ligne de recommandation par profil**.
   * **Pourriez-vous m'expliquer comment vous souhaiteriez que les résultats finaux soient structurés dans l'e-mail pour ces tests ?** Devons-nous afficher toutes les recommandations ? Seulement celles du "Répondant" ? Faut-il les regrouper par Axe ? Votre réponse guidera la restructuration de ces onglets.
4. **Action Finale :** Une fois les données prêtes, déployer les nouvelles versions des scripts Logique\_Universel.gs et Utilities.gs.