Absolument. Voici la synthèse complète de notre session, prête à être sauvegardée. Elle intègre tous les échanges, les décisions, les corrections de code, et l'architecture mise à jour, conformément à votre modèle.

Markdown

---

Projet: Usine à Tests

Date: 2025-07-11

Version: 17.0

---

### 1. Résumé des Échanges du Jour

La session avait pour objectif d'implémenter de nouveaux modes de calcul dans le moteur universel V2.

Nous avons débuté avec l'ambition de créer un mode `QCU\_PONDERE`, mais nous avons rapidement conclu qu'il s'agissait d'une généralisation du mode `QCU\_CAT` existant. La décision stratégique a été de \*\*fusionner les deux en un unique mode `QCU\_CAT` unifié\*\*, capable de gérer des options avec une valeur par défaut de 1 (texte simple) ou une valeur pondérée (objet JSON).

Ce développement a initié un cycle itératif de \*\*Développement → Test → Bug → Correction\*\* qui a considérablement renforcé l'architecture :

1. \*\*Développement de `QCU\_CAT` unifié\*\* : La logique de calcul dans `Logique\_Universel.gs` a été mise à jour.

2. \*\*Bug #1 (Génération Formulaire)\*\* : Le Moteur ne savait pas afficher les options en format texte simple. Nous avons corrigé `creerItemFormulaire` dans `Utils V2.gs`.

3. \*\*Développement de `QCU\_DIRECT`\*\* : Nous avons défini la logique d'affectation directe et l'avons implémentée dans `Logique\_Universel.gs`.

4. \*\*Bug #2 (Erreur JSON)\*\* : Le Moteur a généré une erreur `JSON sans options` car il ne gérait pas les cas où le JSON ne contient pas de liste d'options. Nous avons refactorisé `creerItemFormulaire` pour le rendre plus intelligent.

5. \*\*Développement de `QRM\_CAT`\*\* : Nous avons défini la logique pour les questions à réponses multiples.

6. \*\*Bug #3 (Mauvais Type de Question)\*\* : Le Moteur créait des questions à choix unique (radio) au lieu de cases à cocher. Nous avons finalisé `creerItemFormulaire` pour qu'il choisisse le bon type d'item (`addCheckboxItem` vs `addMultipleChoiceItem`) en fonction du mode (`QRM` vs `QCU`).

La session s'est conclue par un \*\*test final entièrement réussi\*\*, validant le bon fonctionnement simultané des trois modes implémentés.

### 2. État Actuel du Projet

\* \*\*Tâches réalisées :\*\*

\* Implémentation et validation de 3 modes de calcul V2 : \*\*`QCU\_CAT` (unifié), `QCU\_DIRECT`, `QRM\_CAT`\*\*.

\* Correction de 3 bugs majeurs dans le Moteur de génération (`Utils V2.gs`), le rendant plus robuste et flexible.

\* Validation de la cohérence de la chaîne complète V2 (BDD → Moteur → Formulaire → Kit → Calcul).

\* \*\*Tâches en cours :\*\* Aucune.

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

\* Développer un nouveau mode de calcul (ex: un autre type de `QRM` ou `LIKERT`).

\* OU construire le script utilitaire de migration des tests V1 vers le format V2.

### 3. Contexte Historique du Projet

Le projet vise à créer une "Usine à Tests" flexible et entièrement automatisée. Initialement (V1), chaque type de test possédait son propre script de logique de calcul (ex: `Logique\_Couleurs.gs`). La stratégie V2 a introduit un \*\*moteur de calcul universel\*\* (`Logique\_Universel.gs`) capable de traiter n'importe quel type de test via un système de modes et de paramètres JSON.

La session d'aujourd'hui marque la \*\*première implémentation concrète et la validation de cette stratégie V2\*\*. Nous avons transformé le concept de moteur universel en une réalité fonctionnelle en y intégrant ses trois premiers modes de calcul.

### 4. Orientations Stratégiques

L'objectif à long terme reste l'automatisation et la flexibilité maximales. Les échanges du jour ont renforcé cette vision :

\* \*\*Simplification & Flexibilité\*\* : La fusion de `QCU\_PONDERE` dans `QCU\_CAT` simplifie la configuration des tests tout en augmentant la puissance du mode.

\* \*\*Robustesse de l'Architecture\*\* : Le cycle de débogage intensif a permis de "blinder" l'architecture, notamment la fonction `creerItemFormulaire` qui est un point de contact essentiel entre la BDD et les formulaires générés.

\* \*\*Validation du Concept\*\* : Le succès de l'implémentation de trois modes de logique très différente prouve la viabilité et l'extensibilité du moteur universel.

### 5. Derniers Scripts et Ressources Clés

Voici les versions finales et validées des deux fichiers de script que nous avons modifiés aujourd'hui.

#### `Logique\_Universel.gs` (Version 2.2 - Projet Kit V2)

```javascript

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

// == FICHIER : Logique\_Universel.gs

// == VERSION : 2.2 (Implémentation du mode QRM\_CAT)

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

// == configuré dans la BDD selon les modes de 'ref\_Modes\_Traitement'.

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

function calculerResultats(reponsesUtilisateur, langueCible, config) {

let resultats = {

profilFinal: "",

titreProfil: "",

descriptionProfil: "",

scoresData: {},

sousTotauxParMode: {}

};

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

if (!questionsMap || Object.keys(questionsMap).length === 0) {

Logger.log("Aucune question n'a pu être chargée. Arrêt du traitement.");

return {};

}

for (const titreQuestion in reponsesUtilisateur) {

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

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

const questionConfig = questionsMap[idQuestion];

if (questionConfig) {

const reponse = reponsesUtilisateur[titreQuestion][0];

const mode = questionConfig.mode;

const parametres = questionConfig.parametres;

\_aiguillerCalcul(mode, reponse, parametres, resultats);

}

}

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

return 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\_TOUTOURIEN': \_traiterQRM\_TOUTOURIEN(reponse, parametres, resultats); break;

case 'QRM\_PARTIEL\_PEN': \_traiterQRM\_PARTIEL\_PEN(reponse, parametres, resultats); break;

case 'QRM\_CAT': \_traiterQRM\_CAT(reponse, parametres, resultats); break;

case 'LIKERT\_PONDERE': \_traiterLIKERT\_PONDERE(reponse, parametres, resultats); break;

case 'LIKERT\_CAT': \_traiterLIKERT\_CAT(reponse, parametres, resultats); break;

case 'CHOIXFORCE\_CAT': \_traiterCHOIXFORCE\_CAT(reponse, parametres, resultats); break;

case 'SJT\_RANG': \_traiterSJT\_RANG(reponse, parametres, resultats); break;

case 'SJT\_NOTE\_DELTA': \_traiterSJT\_NOTE\_DELTA(reponse, parametres, resultats); break;

case 'MATRICE\_CAT': \_traiterMATRICE\_CAT(reponse, parametres, resultats); break;

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

}

}

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

if (!parametres || !parametres.profil) {

Logger.log(`ERREUR QCU\_DIRECT: Paramètres invalides. La clé "profil" est requise.`);

return;

}

const profil = parametres.profil;

resultats.scoresData[profil] = reponseUtilisateur;

Logger.log(`QCU\_DIRECT: Affectation de la valeur '${reponseUtilisateur}' au profil '${profil}'.`);

}

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

if (!parametres || !parametres.profil || !parametres.options || !Array.isArray(parametres.options)) {

Logger.log(`ERREUR QCU\_CAT: Paramètres invalides. "profil" et "options" sont requis.`);

return;

}

const profil = parametres.profil;

let valeur = null;

for (const opt of parametres.options) {

if (typeof opt === 'object' && opt !== null && opt.libelle === reponseUtilisateur) {

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

break;

} else if (typeof opt === 'string' && opt === reponseUtilisateur) {

valeur = 1;

break;

}

}

if (valeur !== null) {

const mode = 'QCU\_CAT';

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

if (!resultats.sousTotauxParMode[mode]) {

resultats.sousTotauxParMode[mode] = {};

}

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

Logger.log(`QCU\_CAT: Ajout de ${valeur} points au profil '${profil}'. Nouveau score: ${resultats.scoresData[profil]}`);

} else {

Logger.log(`QCU\_CAT: La réponse '${reponseUtilisateur}' n'a pas été trouvée dans les options pour le profil '${profil}'.`);

}

}

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

if (!reponseUtilisateur || !parametres || !parametres.options || !Array.isArray(parametres.options)) {

Logger.log(`ERREUR QRM\_CAT: Paramètres ou réponse manquants/invalides.`);

return;

}

const reponsesArray = reponseUtilisateur.split(', ').map(reponse => reponse.trim());

reponsesArray.forEach(reponse => {

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

if (optionTrouvee) {

if (optionTrouvee.profil && typeof optionTrouvee.valeur === 'number') {

const profil = optionTrouvee.profil;

const valeur = optionTrouvee.valeur;

const mode = 'QRM\_CAT';

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

if (!resultats.sousTotauxParMode[mode]) {

resultats.sousTotauxParMode[mode] = {};

}

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

Logger.log(`QRM\_CAT: Ajout de ${valeur} points au profil '${profil}' pour la réponse '${reponse}'.`);

} else {

Logger.log(`QRM\_CAT: L'option '${reponse}' a été trouvée mais n'a pas de profil/valeur valide.`);

}

} else {

Logger.log(`QRM\_CAT: La réponse cochée '${reponse}' n'a pas été trouvée dans les options des paramètres.`);

}

});

}

function \_traiterQRM\_TOUTOURIEN(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterQRM\_PARTIEL\_PEN(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterLIKERT\_PONDERE(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterLIKERT\_CAT(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterCHOIXFORCE\_CAT(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterSJT\_RANG(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterSJT\_NOTE\_DELTA(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_traiterMATRICE\_CAT(reponse, parametres, resultats) { /\* Logique à coder \*/ }

function \_chargerQuestions(typeTest, langue) {

try {

const systemIds = getSystemIds();

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

const nomFeuille = `Questions\_${typeTest}\_${langue}`;

const sheet = bdd.getSheetByName(nomFeuille);

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

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

const headers = data.shift();

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

const modeCol = headers.indexOf('TypeQuestion');

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

if(idCol === -1 || modeCol === -1 || paramsCol === -1) { throw new Error("Colonnes requises manquantes dans " + nomFeuille); }

const questionsMap = {};

data.forEach(row => {

const id = row[idCol];

if (id) {

try {

if(row[modeCol] && row[paramsCol]) {

questionsMap[id] = { id: id, mode: row[modeCol], parametres: JSON.parse(row[paramsCol]) };

}

} catch (e) { Logger.log(`Erreur parsing JSON ID '${id}': ${e.message}`); }

}

});

Logger.log(`${Object.keys(questionsMap).length} questions chargées pour '${typeTest}' en '${langue}'.`);

return questionsMap;

} catch (e) {

Logger.log("Erreur critique chargement questions: " + e.message);

return null;

}

}

**Utils V2.gs (Version 2.4 - Projet Moteur V2)**

JavaScript

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

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

// RÔLE : Fonctions utilitaires pour le Moteur.

// VERSION : 2.4 - Différenciation QCU/QRM pour création (radio/checkbox)

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

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

function getSystemIds() {

try {

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

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

if (!idSheet) { throw new Error("'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) { throw new Error("Impossible de charger les ID système: " + e.message); }

}

function getConfigurationFromRow(rowIndex) {

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

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

if (!sheet) { throw new Error("'Paramètres Généraux' est introuvable."); }

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

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

const config = {};

headers.forEach((header, i) => { if (header) { config[header] = rowValues[i]; } });

return config;

}

function getLangueFullName(code) {

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

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

}

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

let params = null;

let choices = [];

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

try {

params = JSON.parse(paramsJSONString);

} catch (e) {

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

return;

}

}

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

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

} else if (optionsString) {

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

}

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

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

if (choices.length > 0) {

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

} else {

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

}

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

if (choices.length > 0) {

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

} else {

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

}

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

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

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

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

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

if (libelles.length === 2) {

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

}

} else {

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

}

} else {

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

}

}

**6. Actions Recommandées / Prochaines Étapes**

* **Action Immédiate :** Aucune. Le projet est dans un état stable et les fonctionnalités développées aujourd'hui sont validées.
* **Pour la prochaine session :** Nous devrons choisir notre prochain axe de développement :
  1. **Option A :** Continuer l'implémentation de nouveaux modes de calcul (QRM\_TOUTOURIEN, LIKERT\_PONDERE, etc.).
  2. **Option B :** Développer l'outil de migration pour convertir les tests V1 au format V2.