### Projet : Usine à Tests - Principes d'Architecture V2

Ce document formalise les leçons tirées du débogage du système V1 (logique par script) et les traduit en principes directeurs pour le développement du moteur universel V2 (logique par JSON).

#### Leçon n°1 : La Fragilité des "Conventions Implicites"

* **Problème V1 :** Le système entier reposait sur le fait que le script "savait" que la colonne de l'e-mail s'appelait Adresse Email. Quand ce nom a changé ou que la colonne est devenue vide, tout s'est effondré. Nous avons perdu un temps précieux à chercher la bonne clé (Adresse Email, Adresse e-mail, etc.).
* **Principe V2 : "Zéro Convention Implicite".** Le moteur V2 ne doit jamais "deviner" ou "supposer" le nom d'un champ critique. Toute information dont il a besoin doit être explicitement définie dans sa configuration.
* **Capitalisation pour V2 :**
  1. **Le Moteur V2 ne doit plus ajouter de questions en dur.** Fini les form.addTextItem().setTitle("Votre courriel de contact").
  2. **Standardisation dans la BDD :** Nous devons créer des "blocs de questions" standards dans la base de données. Par exemple, une question avec ID="META\_NOM\_COMPLET" et une autre avec ID="META\_EMAIL".
  3. **Configuration Explicite :** Dans la feuille Paramètres Généraux, une nouvelle colonne Blocs\_Meta\_A\_Inclure pourrait lister les ID à ajouter au début de chaque formulaire (ex: META\_NOM\_COMPLET, META\_EMAIL). Le Moteur lit cette liste et va chercher les questions correspondantes dans la BDD pour les créer.

#### Leçon n°2 : La Source de Données Unique

* **Problème V1 :** Nous avions un conflit entre deux sources de vérité pour l'e-mail : la collecte automatique de Google et la question manuelle. Cela a créé une confusion totale et des bugs imprévisibles.
* **Principe V2 : "La Source Unique de Vérité est le JSON".** Pour une question donnée, la seule source d'information qui fait foi est son objet Paramètres (JSON) dans la base de données.
* **Capitalisation pour V2 :**
  1. La fonction creerItemFormulaire de la V2 doit être rendue encore plus intelligente. Elle ne doit plus lire que le JSON. Si les options d'une question sont dans le JSON ("options": [...]), elle les utilise. Si le JSON est vide, elle ne fait rien ou crée une question ouverte, mais elle ne doit plus jamais essayer de lire une autre colonne comme Options dans la feuille de calcul.
  2. Cela simplifie radicalement la configuration : **tout ce qui concerne une question est dans une seule cellule : son JSON.**

#### Leçon n°3 : La Robustesse face à la Configuration

* **Problème V1 :** Le script plantait brutalement si une colonne de configuration (Repondant\_Email\_Actif) n'existait pas.
* **Principe V2 : "Le Code doit être plus Intelligent que la Configuration".** Le moteur universel V2 doit être conçu pour ne jamais planter à cause d'une configuration manquante ou malformée. Il doit anticiper les erreurs.
* **Capitalisation pour V2 :**
  1. **Vérifications Systématiques :** Chaque fonction de traitement dans Logique\_Universel.gs (ex: \_traiterQCU\_CAT) doit commencer par un bloc de validation qui vérifie la présence de toutes les clés nécessaires dans l'objet parametres.  
     function \_traiterQCU\_CAT(reponse, parametres, resultats) {  
      // Vérification de robustesse  
      if (!parametres || !parametres.profil || !parametres.options) {  
      Logger.log("ERREUR QCU\_CAT: Paramètres invalides pour la réponse " + reponse);  
      return; // On arrête le traitement pour cette question, mais on ne plante pas le script.  
      }  
      // ... reste de la logique ...  
     }
  2. **Gestion des Valeurs par Défaut :** Le code doit utiliser des valeurs par défaut logiques. Par exemple, dans Utilities.gs, la ligne const niveau = config.Repondant\_Contenu || "Niveau 1 : Simple"; est un excellent exemple de ce principe.

**Projet : Usine à Tests - Leçon d'Architecture n°4**

**Leçon n°4 : Isoler et Ne Jamais Faire Confiance Aveuglément aux Services Externes**

* Problème Rencontré : Le bug le plus complexe que nous ayons résolu n'était pas dans notre logique, mais dans le comportement imprévisible du service MailApp de Google. Le log nous disait qu'un e-mail était envoyé à une adresse A, alors qu'en réalité, il était envoyé à une adresse B. Le service échouait silencieusement ou se comportait de manière inattendue. Plus tard, il nous a bloqués à cause de quotas dépassés.
* Principe V2 : "Le Moteur est une Boîte Noire Fiable dans un Monde Incertain". L'architecture du moteur V2 doit considérer que tout service externe (Google Mail, Google Drive, etc.) est potentiellement peu fiable. Notre code doit être conçu pour anticiper et survivre à ces défaillances, et pour nous dire la vérité même quand le service externe "ment".
* Capitalisation pour V2 :
  1. Généraliser la Séparation des Appels : Le principe "diviser pour régner" doit être appliqué partout. Si le moteur V2 doit un jour effectuer plusieurs actions critiques (ex: envoyer un e-mail, créer un fichier, mettre à jour une cellule), chaque action doit être dans son propre bloc try...catch pour que l'échec de l'une n'entraîne pas l'échec des autres.
  2. Journalisation Défensive : Les logs doivent refléter l'intention du script, pas seulement le résultat de l'appel.
     + Mauvais log : Email envoyé.
     + Bon log : Tentative d'envoi à l'adresse X... suivi de SUCCÈS ou ERREUR. Cela nous permet de voir ce que le script *voulait* faire, même si le service a fait autre chose.
  3. Gestion Explicite des Quotas : Pour des actions critiques comme l'envoi d'e-mails, la V2 pourrait inclure une gestion des quotas. Par exemple, si un grand nombre d'e-mails doit être envoyé, le script pourrait les mettre en file d'attente et les envoyer par petits lots, avec des pauses entre chaque lot, pour éviter de dépasser les limites de Google.

Cette dernière leçon est cruciale. Elle nous fait passer d'un script qui "fonctionne" à un système de qualité professionnelle, capable de résister aux aléas du monde réel.

### Plan d'Action Recommandé

1. **Mettre à jour la Documentation :** Intégrer ces trois principes dans votre document Documentation de l'Architecture de l'Usine à Tests.
2. **Refactoriser le Moteur V2 :** Modifier la fonction de création de formulaire du Moteur V2 pour qu'elle n'ajoute plus de questions en dur, mais se base sur une configuration dans Paramètres Généraux (Leçon n°1).
3. **Renforcer la Logique V2 :** Avant de développer de nouveaux modes de calcul, passer en revue les fonctions existantes dans Logique\_Universel.gs et y ajouter les blocs de validation systématique (Leçon n°3).

En appliquant ces principes, votre V2 ne sera pas seulement plus puissante, elle sera infiniment plus fiable et facile à maintenir que la V1.