Wiki : Script Frontend StockAV (script.js)

1. Introduction

Ce document décrit le fonctionnement du fichier script.js, le cœur de la logique frontend pour l'application **StockAV**. Cette application web permet la gestion de stock de composants électroniques, incluant la recherche, la modification des quantités, l'administration des références et des catégories, ainsi que l'import/export de données. Elle s'appuie sur **Supabase** pour la base de données, l'authentification et les fonctions serveur (Edge Functions).

2. Concepts et Technologies Clés

* **Supabase:** Utilisé comme Backend-as-a-Service.
  + supabase: Variable client pour interagir avec l'API Supabase.
  + **Authentification:** Gérée via supabase.auth (signInWithPassword, signOut, onAuthStateChange). Utilise un système de code + mot de passe mappé sur une adresse email fictive (FAKE\_EMAIL\_DOMAIN).
  + **Base de données:** Interaction avec les tables inventory, categories, et logs via supabase.from('table\_name').select()/.insert()/.update()/.delete()/.upsert().
  + **Edge Functions:** Appel de fonctions serveur (ex: ai-component-info) pour des logiques spécifiques comme la recherche d'équivalents IA via supabase.functions.invoke().
* **JavaScript Asynchrone:** Utilisation intensive de async/await pour gérer les appels réseaux vers Supabase.
* **Manipulation du DOM:** Mise à jour dynamique de l'interface utilisateur en sélectionnant des éléments HTML (via document.getElementById, document.querySelectorAll) et en modifiant leur contenu (textContent, innerHTML), style (style.display) ou classes (classList).
* **Gestion d'État:** Des variables globales suivent l'état de l'application :
  + currentUser, currentUserCode: Informations sur l'utilisateur connecté.
  + activeSession: La session Supabase actuelle.
  + categoriesCache: Cache local des catégories pour la performance.
  + currentInventoryPage, currentLogPage: Suivi de la pagination.
  + conversationState: État de la conversation du chatbot.
* **Gestion Événementielle:** Utilisation de addEventListener pour réagir aux actions de l'utilisateur (clics, saisie clavier).
* **PapaParse:** Librairie externe (chargée via CDN) utilisée pour l'analyse syntaxique (parsing) et la génération de fichiers CSV (handleImportInventoryCSV, handleExportInventoryCSV).
* **Mode Strict:** "use strict"; activé pour un code plus propre et moins sujet aux erreurs silencieuses.
* **Exécution Différée:** Le code principal est encapsulé dans document.addEventListener('DOMContentLoaded', () => { ... }); pour s'assurer que le HTML est chargé avant l'exécution.

3. Structure Générale et Initialisation

* **Initialisation Globale (initializeApp()):**
  + Fonction principale appelée après le chargement du DOM.
  + Vérifie la présence des éléments HTML essentiels. S'arrête avec une erreur si un élément manque.
  + Attache tous les event listeners nécessaires aux boutons, inputs, tables, etc. C'est ici que les fonctions de callback (comme handleLogin, handleUserInput, handleInventoryRowClick, setActiveView) sont liées aux interactions utilisateur.
  + Démarre l'écouteur d'état d'authentification (setupAuthListener).
  + Initialise l'affichage 7 segments (updateSevenSegmentDisplay).
* **Vérifications Initiales:** Le script vérifie au début la présence des clés Supabase et des librairies (supabase-js, PapaParse).

4. Fonctionnalités Principales

4.1. Authentification

* **Connexion (handleLogin()):**
  + Récupère le code et le mot de passe.
  + Construit l'email fictif.
  + Appelle supabase.auth.signInWithPassword().
  + Gère les erreurs (identifiants incorrects, autres erreurs).
  + Charge les catégories (getCategories()) si nécessaire après succès.
* **Déconnexion (handleLogout()):**
  + Appelle supabase.auth.signOut().
  + Réinitialise l'état (cache catégories, affichage 7 segments).
* **Suivi de l'État (setupAuthListener()):**
  + Vérifie la session initiale avec supabase.auth.getSession().
  + S'abonne aux changements d'état (supabase.auth.onAuthStateChange).
  + Appelle handleUserConnected() ou handleUserDisconnected() en fonction des événements (SIGNED\_IN, SIGNED\_OUT).
* **Mise à Jour UI (handleUserConnected(), handleUserDisconnected()):**
  + Met à jour les variables currentUser et currentUserCode.
  + Ajoute/retire la classe user-logged-in au body.
  + Affiche/masque les zones de login ou d'info utilisateur.
  + Affiche/masque les boutons protégés (protectedButtons).
  + **Restriction Spécifique:** Le bouton Paramètres (settingsTabButton) est affiché uniquement si currentUserCode est 'zine'.
  + Nettoie les données des vues protégées (clearProtectedViewData()) à la déconnexion.
  + Redirige vers la vue de recherche si l'utilisateur est déconnecté alors qu'il était sur une vue protégée.

4.2. Navigation

* **setActiveView():**
  + Fonction centrale pour afficher une vue (searchView, inventoryView, etc.) et masquer les autres.
  + Met en évidence le bouton de navigation correspondant (nav-button).
  + Vérifie si l'utilisateur est connecté avant d'afficher une vue protégée.
  + **Restriction Spécifique:** Empêche l'accès à settingsView si l'utilisateur connecté n'est pas 'zine'.
  + Déclenche le chargement des données nécessaires pour la vue activée (ex: displayInventory(), displayLog()).

4.3. Vue Inventaire

* **Affichage (displayInventory()):**
  + Récupère les données de la table inventory via supabase.from().select() avec pagination (range()) et tri (order()).
  + Prend en compte les filtres (currentInventoryFilters).
  + Utilise le categoriesCache pour afficher le nom des catégories (via categoryNameMap).
  + Génère les lignes du tableau (inventoryTableBody).
  + Affiche un indicateur de stock (stock-indicator) basé sur getStockStatus().
  + Met à jour les informations et l'état des boutons de pagination (inventoryPageInfo, inventoryPrevPageButton, inventoryNextPageButton).
* **Filtrage:**
  + populateInventoryFilters(): Remplit le select des catégories depuis categoriesCache.
  + applyInventoryFilterButton listener: Met à jour currentInventoryFilters et appelle displayInventory(1).
* **Modification Quantité (Modale):**
  + handleInventoryRowClick(): Callback pour clic sur une ligne. Récupère les détails complets (y compris attributes) via getStockInfoFromSupabase() et appelle showQuantityModal().
  + showQuantityModal(): Affiche la modale (quantityChangeModal, modalOverlay), pré-remplit les infos (réf, qté). Crée et affiche les **badges d'attributs** (attribute-badge) dans #modal-attributes-list en utilisant getBadgeClassForKey() pour la couleur.
  + hideQuantityModal(): Masque la modale et réinitialise son état.
  + updateModalButtonStates(): Active/désactive les boutons +/- et Confirmer.
  + Listeners sur modalDecreaseButton, modalIncreaseButton, modalConfirmButton, modalCancelButton: Gèrent l'incrémentation/décrémentation et la confirmation/annulation.
  + Le bouton Confirmer appelle updateStockInSupabase() puis displayInventory() si succès.

4.4. Vue Historique (Log)

* **Affichage (displayLog()):**
  + Similaire à displayInventory, mais récupère les données de la table logs, triées par date décroissante.
  + Formate le timestamp (formatLogTimestamp()).
  + Gère la pagination (logPageInfo, etc.).
* **Enregistrement (addLogEntry()):**
  + Appelée après chaque modification de stock réussie (updateStockInSupabase).
  + Insère une nouvelle ligne dans la table logs via supabase.from().insert().

4.5. Vue Administration

* **Chargement (loadAdminData()):** Appelle loadCategoriesAdmin() et réinitialise les formulaires.
* **Gestion Catégories:**
  + loadCategoriesAdmin(): Affiche la liste des catégories (categoryList) et remplit le select (componentCategorySelectAdmin) en utilisant getCategories().
  + addCategoryEventListeners(): Ajoute les listeners pour les boutons Modifier/Supprimer de la liste et pour la soumission du formulaire (categoryForm).
    - Modification: Pré-remplit le formulaire.
    - Suppression: Appelle supabase.from().delete() avec confirmation. Gère les erreurs de clé étrangère.
    - Ajout/Sauvegarde: Appelle supabase.from().insert() ou .update() selon le contexte (editingId). Gère les erreurs de nom dupliqué (23505).
  + resetCategoryForm(): Vide le formulaire de catégorie.
* **Gestion Stock:**
  + addComponentCategorySelectListener(): Affiche/masque et génère les champs d'attributs spécifiques (specificAttributesDiv) quand une catégorie est sélectionnée dans le formulaire de stock.
  + addStockEventListeners(): Ajoute les listeners pour les boutons du formulaire de stock.
    - checkStockButton: Appelle getStockInfoFromSupabase(). Pré-remplit le formulaire si la référence existe, sinon prépare le formulaire pour un ajout. Déclenche l'affichage des attributs spécifiques si applicable.
    - updateQuantityButton: Appelle updateStockInSupabase() pour un ajustement rapide.
    - stockForm (submit): Valide les données. Collecte les attributs spécifiques. Appelle supabase.from().upsert() pour ajouter ou mettre à jour le composant complet.
  + resetStockForm(): Vide le formulaire de stock.

4.6. Vue Paramètres

* **Chargement (loadSettingsData()):** Réinitialise les zones de feedback.
* **Export:**
  + handleExportInventoryCSV(): Récupère tout l'inventaire, utilise categoriesCache pour les noms, génère le CSV avec Papa.unparse(), et déclenche le téléchargement via downloadFile().
  + handleExportLogTXT(): Récupère tous les logs, formate en texte simple, et déclenche le téléchargement via downloadFile().
* **Import:**
  + handleImportInventoryCSV():
    - Lit le fichier sélectionné (importCsvFileInput) avec Papa.parse().
    - Lit le **mode d'importation** choisi par l'utilisateur (radios import-mode).
    - Demande une **confirmation** si le mode "Écraser" (overwrite) est sélectionné.
    - Valide les en-têtes et chaque ligne du CSV (réf, qté, catégorie via categoryMap, format attributs JSON).
    - Si mode "Écraser", exécute supabase.from('inventory').delete() **avant** l'upsert (nécessite politique RLS DELETE).
    - Exécute supabase.from('inventory').upsert() avec les données validées (itemsToUpsert).
    - Affiche des messages de feedback (showSettingsFeedback()).
  + resetImportState(): Réinitialise le formulaire d'import.
* addSettingsEventListeners(): Attache les listeners aux boutons d'export/import.

4.7. Logique Chat (Vue Recherche)

* **Interface (addMessageToChat(), displayWelcomeMessage()):** Gère l'affichage des messages utilisateur et IA dans responseOutputChat, incluant l'effet "typing" et la gestion de l'historique (chatHistory).
* **Entrée Utilisateur (handleUserInput()):**
  + Appelée lors du clic sur searchButtonChat or Entrée dans componentInputChat.
  + Vérifie si le chat attend une confirmation de quantité (conversationState.awaitingQuantityConfirmation). Si oui, appelle handleQuantityResponse().
  + Sinon, tente d'extraire une référence (extractReference()).
  + Si une référence est trouvée, appelle checkComponentWithAI().
  + Gère les cas où l'entrée n'est pas comprise.
* **Extraction Référence (extractReference()):** Utilise une série d'expressions régulières (patterns) pour tenter d'identifier une référence de composant dans le texte utilisateur.
* **Vérification & IA (checkComponentWithAI()):**
  + Fonction principale du chat.
  + Vérifie le stock local de la référence originale via getStockInfoFromSupabase().
  + Appelle la fonction Edge Supabase ai-component-info via supabase.functions.invoke() pour obtenir des équivalents.
  + Vérifie le stock local des équivalents reçus.
  + Construit une réponse HTML (responseHTML) incluant le statut de l'original, la liste des équivalents (avec leur statut de stock, des liens externes via provideExternalLinksHTML(), et des boutons "Prendre" si dispo et connecté).
  + Gère l'état de la conversation (conversationState.awaitingEquivalentChoice).
* **Interaction (responseOutputChat listener, promptLoginBeforeAction()):**
  + Détecte les clics sur les boutons "Prendre".
  + Vérifie si l'utilisateur est connecté avant de continuer (promptLoginBeforeAction).
  + Met à jour l'état (conversationState) pour attendre la quantité.
  + Appelle addMessageToChat() pour demander la quantité.
* **Confirmation Quantité (handleQuantityResponse()):**
  + Parse la quantité entrée par l'utilisateur.
  + Valide la quantité par rapport au stock disponible (conversationState.availableQuantity).
  + Appelle updateStockInSupabase() avec une quantité négative.
  + Affiche le résultat de l'opération.
  + Réinitialise l'état (resetConversationState()).

4.8. Affichage 7 Segments

* **updateSevenSegmentDisplay():**
  + Met à jour l'affichage (sevenSegmentDisplay) en fonction de la valeur lastDisplayedDrawer.
  + Est cachée (display-off) si l'utilisateur n'est pas connecté ou si aucune valeur n'est à afficher.
  + Utilise segmentMap pour déterminer quels segments (segment-a à segment-g) allumer (addClass('on')) pour chaque caractère.

5. Dépendances Externes

* **Supabase JS Client:** @supabase/supabase-js (via CDN)
* **PapaParse:** Pour le parsing CSV (via CDN)