Mémo narratif – Fichiers JS (Projet 3)

script.js

Ce fichier gère l'affichage principal de la galerie et l'interface entre le mode visiteur et le mode administrateur.

- Au début, on vérifie si un **token** existe dans le localStorage. → Si oui : on affiche la barre d'administration, le bouton logout, et les boutons "éditer". → Sinon : on reste en mode visiteur simple.

- Ensuite, on installe un **événement sur le bouton logout** : quand on clique, le token est supprimé et la page est rechargée pour repasser en mode visiteur.

- On fait ensuite un **fetch vers l'API /works** pour récupérer tous les projets depuis la base de données. → Ces projets sont stockés dans une variable globale `allProjects`. → Puis on appelle la fonction `displayProjects()` qui construit le HTML de la galerie en insérant dynamiquement chaque projet avec son image et son titre.

- On fait aussi un **fetch vers l'API /categories** pour récupérer la liste des catégories. → Avec ça, on crée dynamiquement les boutons de filtre ("Tous", "Objets", "Appartements", etc.). → Chaque bouton reçoit un écouteur d'événement : quand on clique, ça filtre la galerie en ne montrant que les projets de la catégorie choisie.

- Enfin, la fonction `displayProjects()` se charge d'effacer la galerie puis de réinsérer les projets demandés (soit tous, soit filtrés).

■ En résumé : `script.js` gère le mode admin/visiteur, la récupération des projets, la construction de la galerie, et le filtrage par catégories.

login.js

Ce fichier gère la page de connexion de l'administrateur.

- On installe un **événement sur le formulaire de login**. Quand on le soumet : → On récupère l'email et le mot de passe saisis. → On fait un `fetch` POST vers l'API `/users/login` avec ces identifiants.

- Si l'API répond avec succès (code 200), elle renvoie un **token** : → On le stocke dans le `localStorage`. → Puis on redirige l'utilisateur vers `index.html` en mode administrateur.

- Si l'API répond avec une erreur (401 ou 404), on affiche un **message d'erreur** : "Email ou mot de passe incorrect". - Si jamais le serveur est injoignable, on affiche : "Une erreur est survenue".

■ En résumé : `login.js` vérifie les identifiants de l'admin, stocke le token si c'est correct, et permet d'entrer en mode administrateur.

modals.js

Ce fichier gère toutes les fenêtres modales de l'administration (galerie modifiable et ajout de photos).

- On commence par sélectionner les boutons et les éléments modaux (`.modal`, `.modal-add`, boutons "fermer", etc.).

- Quand on clique sur "éditer" : → La modale s'affiche. → On appelle `displayWorksInModals()` qui récupère la liste des projets et les affiche dans la modale, chacun avec son image et une icône poubelle.

- Chaque icône poubelle est liée à un **événement de suppression** : → On demande confirmation avec `confirm()`. → Si c'est validé : on envoie une requête DELETE vers l'API `/works/:id`. → En cas de succès : l'image est supprimée de la modale ET la galerie principale est mise à jour.

- Un autre bouton permet d’ouvrir la modale d’ajout de photo. → On choisit une image depuis son ordinateur. → On affiche un **aperçu** avec `FileReader`. → On choisit aussi un titre et une catégorie.

- À la soumission du formulaire d’ajout : → On prépare un `FormData` avec l’image, le titre et la catégorie. → On fait un POST vers `/works` avec le token admin. → En cas de succès : la photo est ajoutée, le formulaire est réinitialisé, la modale se ferme, et la galerie est mise à jour.

- Des vérifications de formulaire sont faites en temps réel (par ex. bouton "Valider" activé uniquement si tous les champs sont remplis).

■ En résumé : `modals.js` permet à l’admin d’afficher la galerie dans une modale, de supprimer des projets existants, et d’ajouter de nouveaux projets avec un aperçu avant validation.