5 - Gestion des formulaires et ajout de données

🤔 Concepts à comprendre

Formulaires HTML et JavaScript

Traditionnellement, un formulaire HTML soumet des données et recharge la page. En développement moderne (SPA - Single Page Application), on intercepte cette soumission pour :

- Empêcher le rechargement
- Valider les données
- Envoyer les données via Fetch
- Mettre à jour l'interface sans recharger

```javascript
form.addEventListener('submit', (event) => { event.preventDefault(); // Empêche le comportement par défaut // Votre code ici });
```

Récupération des valeurs de formulaire

Méthode 1 : FormData API

```JavaScript
const formData = new FormData(form);
const data =
Object.fromEntries(formData);
```

Méthode 2 : Sélection manuelle

```JavaScript
const brand =
document.getElementById('brand').valu
const model =
document.getElementById('model').valu
// etc.
```

POST request avec Fetch

Pour envoyer des données à l'API :

```JavaScript
fetch(url, { method: 'POST',
headers: { 'Content-Type':
'application/json' }, body:
JSON.stringify(data) })
```

Points importants :

- `method: 'POST'` : spécifier la méthode HTTP
- `headers` : indiquer qu'on envoie du JSON
- `body` : convertir l'objet JavaScript en chaîne JSON

🔐 Authentification par clé API (IMPORTANT)

Votre API backend utilise un système de **clé API** pour sécuriser l'accès. Cela signifie que toutes vos requêtes qui modifient des données (POST, PUT, DELETE) doivent inclure un header spécial.

Comment fonctionne la clé API ?

Votre backend vérifie la présence d'un header `x-api-key` dans chaque requête. Si cette clé est absente ou invalide, la requête est rejetée avec une erreur 401 (non autorisé) ou 403 (accès refusé).

Clé API à utiliser : `ma-super-cle-api-2024`

⚠️ **ATTENTION** : En production réelle, cette clé devrait être secrète et stockée de manière sécurisée. Pour ce TP, nous l'utilisons en clair pour simplifier.

Code à utiliser pour TOUTES les requêtes POST, PUT, DELETE :

```javascript
fetch(url, { method: 'POST',
headers: { 'Content-Type':
'application/json', 'x-api-key':
'ma-super-cle-api-2025' // ←
OBLIGATOIRE }, body:
JSON.stringify(data) })
```

Exemple complet pour créer une voiture :

```javascript
async function createCar(carData) {
try { const response = await
fetch('https://votre-
api.onrender.com/cars', { method:
'POST', headers: { 'Content-Type':
'application/json', 'x-api-key':
'ma-super-cle-api-2025' }, body:
JSON.stringify(carData) }); if
(!response.ok) { throw new
Error(`Erreur HTTP:
${response.status}`); } return await
response.json(); } catch (error) {
console.error('Erreur lors de la
création:', error); return null; } }
```

Exemple pour supprimer une voiture :

```javascript
async function deleteCar(carId) {
try { const response = await
fetch(`https://votre-
api.onrender.com/cars/${carId}`, {
method: 'DELETE', headers: { 'x-api-
key': 'ma-super-cle-api-2024' // ←
Même pour DELETE } }); if
(!response.ok) { throw new
Error(`Erreur HTTP:
${response.status}`); } return true;
} catch (error) {
console.error('Erreur lors de la
suppression:', error); return false;
} }
```

💡 Conseil : Centraliser la configuration

JavaScript

```
// Dans config.js export const
API_CONFIG = { BASE_URL:
'https://votre-api.onrender.com',
API_KEY: 'ma-super-cle-api-2024' };
// Fonction utilitaire pour les
requêtes authentifiées export async
function fetchWithAuth(url, options
= {}) { const defaultHeaders = {
'Content-Type': 'application/json',
'x-api-key': API_CONFIG.API_KEY };
return fetch(url, { ...options,
headers: { ...defaultHeaders,
...options.headers } }); }
```

Utilisation :

JavaScript

```
import { fetchWithAuth, API_CONFIG }
'./config.js'; // Créer une voiture a
createCar(carData) { const response =
fetchWithAuth(`${API_CONFIG.BASE_URL}
method: 'POST', body: JSON.stringify(
return response.json(); } // Supprime
async function deleteCar(carId) { cor
await
fetchWithAuth(`${API_CONFIG.BASE_URL}
{ method: 'DELETE' }); return respons
```

✅ Tâches à réaliser

4.1 - Créer le formulaire dans le modal

Objectif : Remplacer le contenu placeholder du modal par un formulaire complet pour ajouter une voiture.

Structure HTML à créer (dans le `.modal-body`) :

Vous devez créer un formulaire avec les champs suivants :

- **Brand** (Marque) : input text, requis
- **Model** (Modèle) : input text, requis
- **Year** (Année) : input number, requis (min: 1900, max: année actuelle)
- **Color** (Couleur) : input text, requis
- **Price** (Prix) : input number, requis (min: 0)
- **Mileage** (Kilométrage) : input number, requis (min: 0)
- **Description** : textarea, optionnel
- **Image URL** : input url, optionnel

Classes Bootstrap utiles :

- `mb-3` : margin-bottom
- `form-label` : style des labels
- `form-control` : style des inputs
- `is-invalid` : pour les champs en erreur
- `invalid-feedback` : pour les messages d'erreur

Exemple de structure pour un champ :

```
<div class="mb-3"> <label
for="brand" class="form-
label">Marque *</label> <input
type="text" class="form-control"
id="brand" name="brand" required>
<div class="invalid-feedback">
Veuillez entrer une marque. </div>
</div>
```

Questions à vous poser :

- Faut-il créer ce formulaire en HTML directement ou dynamiquement avec JavaScript ?
- Comment valider que tous les champs requis sont remplis ?
- Comment gérer les types de données (number, text, url) ?

Conseil : Pour ce formulaire, il est plus simple de l'écrire directement en HTML dans `index.html`, dans la section `.modal-body`.

4.2 - Valider les données du formulaire

Objectif : Créer une fonction qui valide les données avant de les envoyer à l'API.

Validations à implémenter :

1. **Champs requis** : brand, model, year, color, price, mileage

2. **Types de données** :
    - Year, price, mileage doivent être des nombres
    - Price et mileage doivent être positifs
    - Year doit être entre 1900 et l'année actuelle
3. **Format** :
    - ImageUrl doit être une URL valide (ou vide)

Fonction de validation suggérée :

JavaScript

```
function validateCarData(data) {
const errors = []; // Vérifier les
champs requis if (!data.brand ||
data.brand.trim() === '') {
errors.push('La marque est
requise'); } // Vérifier le format
de l'année const currentYear = new
Date().getFullYear(); if (data.year
< 1900 || data.year > currentYear) {
errors.push(`L'année doit être entre
1900 et ${currentYear}`); } // ...
autres validations return { isValid:
errors.length === 0, errors: errors
}; }
```

Affichage des erreurs :

Utilisez les classes Bootstrap pour afficher les erreurs :

- Ajouter la classe `is-invalid` aux champs en erreur
- Afficher le message dans un `.invalid-feedback`

Alternative : Afficher toutes les erreurs dans une alerte en haut du modal.

4.3 - Créer une fonction pour envoyer une nouvelle voiture

Objectif : Écrire une fonction asynchrone qui envoie les données à l'API.

Signature suggérée :

```
async function createCar(carData) {
// retourne la voiture créée ou null
en cas d'erreur }
```

Structure de la requête :

```
FONCTION createCar(carData):
ESSAYER: faire une requête POST vers
l'API avec le header Content-Type:
application/json avec les données
converties en JSON dans le body SI
la réponse n'est pas OK: lancer une
erreur avec le statut convertir la
réponse en JSON retourner les
données de la voiture créée EN CAS
D'ERREUR: logger l'erreur
(optionnel) afficher un message à
l'utilisateur retourner null
```

Points d'attention :

- Votre API retourne-t-elle la voiture créée ? (avec son nouvel ID)
- Gère-t-elle les doublons ? Les champs manquants ?
- Quel code de statut retourne-t-elle en cas de succès ? (201 Created, 200 OK ?)

4.4 - Gérer la soumission du formulaire

Objectif : Connecter tous les éléments ensemble.

Flux complet :

1. L'utilisateur remplit le formulaire et clique sur "Enregistrer"
2. Intercepter l'événement submit

3. Récupérer les données du formulaire
4. Valider les données
5. Si invalides : afficher les erreurs et arrêter
6. Si valides : envoyer à l'API
7. Si succès : fermer le modal, rafraîchir la liste, afficher un message de succès
8. Si échec : afficher un message d'erreur

JavaScript

```
async function
handleFormSubmit(event) {
event.preventDefault(); // CRUCIAL !
// 1. Récupérer les données const
form = event.target; const formData =
new FormData(form); const carData =
Object.fromEntries(formData); // 2.
Convertir les types (année, prix,
kilométrage en nombres) carData.year
= parseInt(carData.year);
carData.price =
parseFloat(carData.price);
carData.mileage =
parseInt(carData.mileage); // 3.
Valider const validation =
validateCarData(carData); if
(!validation.isValid) { // Afficher
les erreurs return; } // 4. Envoyer à
l'API const newCar = await
createCar(carData); // 5. Gérer le
résultat if (newCar) { // Succès ! //
Fermer le modal // Rafraîchir la
liste // Réinitialiser le formulaire
} else { // Échec // Afficher une
erreur } } // Attacher l'événement
const form =
document.querySelector('#exampleModal
form');
form.addEventListener('submit',
handleFormSubmit);
```

4.5 - Fermer le modal et rafraîchir la liste

Fermer le modal :

Bootstrap fournit une API JavaScript pour contrôler les modals :

```javascript
const modal =
bootstrap.Modal.getInstance(document.
modal.hide();
```

Ou plus simplement, déclencher un clic sur le bouton de fermeture :

```javascript
document.querySelector('#exampleModal
.btn-close').click();
```

Rafraîchir la liste :

Deux approches :

Approche 1 : Recharger toutes les données

```javascript
const cars = await fetchAllCars();
displayCars(cars);
```

- Simple
- Garantit la cohérence avec l'API

- Mais fait une requête supplémentaire

Approche 2 : Ajouter la nouvelle voiture à la liste existante

```javascript
const container = document.querySelector('.card-cont'); const newCard = createCarCard(newCar); container.insertBefore(newCard, container.firstChild); // Ajouter au début
```

- Plus rapide
- Mais peut ne pas refléter l'ordre exact de l'API

Réflexion : Quelle approche préférez-vous et pourquoi ?

Réinitialiser le formulaire :

```javascript
form.reset(); // Vide tous les champs // Supprimer les classes d'erreur form.querySelectorAll('.is-invalid').forEach(el => { el.classList.remove('is-invalid'); });
```

Heading 3

UX (User Experience)

- **Feedback immédiat** : Validation en temps réel pendant la saisie
- **Messages clairs** : "La marque est requise" plutôt que "Erreur champ 1"
- **Désactiver le bouton pendant l'envoi** : Éviter les doubles soumissions

```javascript
submitButton.disabled = true;
submitButton.textContent = 'Envoi en cours...';
```

Gestion des erreurs réseau

Que faire si :

- L'utilisateur n'a pas de connexion internet ?
- L'API est en maintenance ?
- Le serveur est lent et met 30 secondes à répondre ?

Solutions :

- Timeout sur les requêtes
- Messages d'erreur explicites
- Option de réessayer