3 - Manipulation du DOM et création dynamique de contenu

Création de cartes pour index.html

Approche recommandée

Vous allez créer des éléments HTML en JavaScript pour chaque voiture récupérée de l'API.

Considérations importantes :

- **Performances** : Utilisez `DocumentFragment` ou construisez une chaîne HTML pour minimiser les reflows
- **Accessibilité** : Incluez des attributs `alt` descriptifs pour les images
- **Sécurité** : Évitez `innerHTML` avec des données non validées (risque XSS)

Structure d'une carte

Chaque carte doit contenir :

1. Un lien `<a>` vers `car.html?id={id}`
2. Une image avec l'URL provenant de l'API
3. Le titre (Year + Make + Model)
4. La description (highlights)

5. Un bouton "See more"

Méthodes DOM à utiliser

- `document.createElement()` : créer un élément
- `element.classList.add()` : ajouter des classes CSS
- `element.setAttribute()` : définir des attributs
- `element.textContent` : insérer du texte (sécurisé)
- `element.appendChild()` : attacher un enfant
- `parent.append(...children)` : attacher plusieurs enfants

Remplissage de la page de détail

Extraction de l'ID depuis l'URL

L'URL sera de type : `car.html?id=1`

Techniques possibles :

- `URLSearchParams` : API moderne et recommandée
- `window.location.search` : accès direct à la query string
- Méthodes de parsing manuel (déconseillé)

🤔 Concepts à comprendre

DOM : Le Document Object Model

Le DOM est une représentation en arborescence de votre page HTML. JavaScript peut :

- Lire le contenu
- Modifier le contenu
- Ajouter/supprimer des éléments
- Modifier le style

✅ Tâches à réaliser

3.1 - Créer une fonction pour générer une carte de voiture

Objectif : Écrire une fonction qui prend les données d'une voiture en paramètre et retourne un élément HTML complet (la carte).

Signature de fonction suggérée :

```javascript
function createCarCard(car) { // retourne un élément <article> }
```

Structure HTML à reproduire (référez-vous au HTML existant) :

```HTML
<article class="card shadow-sm"> <a
href="car.html?id=..."> <img
src="..." class="card-img-top"
alt="..."> </a> <div class="card-
body"> <h5 class="card-title">...
</h5> <p class="card-text">...</p>
<a href="car.html?id=..." class="btn
btn-primary">See more</a> </div>
</article>
```

Étapes de réflexion :

1. **Créer l'article :**
   - Quel élément créer en premier ?
   - Quelles classes ajouter ?
2. **Créer le lien image :**
   - Comment créer un élément `<a>` ?
   - Comment définir l'attribut `href` avec l'ID dynamique ?
   - Comment ajouter l'image à l'intérieur ?
3. **Créer l'image :**
   - Attributs nécessaires : `src`, `class`, `alt`
   - Que mettre dans `alt` pour l'accessibilité ?
   - Que faire si `car.imageUrl` est undefined ?

4. **Créer le corps de la carte** :
    - Créer une `div` avec la classe `card-body`
    - Créer le titre (`h5`)
    - Créer la description (`p`)
    - Créer le bouton "See more"
5. **Assembler les éléments** :
    - Utiliser `appendChild()` pour construire la hiérarchie
    - Ordre : article > lien + body ; lien > img ; body > titre + description + bouton

Méthodes DOM utiles :

- `document.createElement(tagName)`
- `element.className = "classes"`
- `element.textContent = "texte"`
- `element.setAttribute(name, value)`
- `element.href = "url"` (pour les liens)
- `parent.appendChild(child)`

Exemple de structure (début de fonction) :

```javascript
function createCarCard(car) { // 1. Créer l'article principal const article = document.createElement('article'); article.className = 'card shadow-sm'; // 2. Créer le lien pour l'image // ...votre code ici // 3. Créer l'image // ...votre code ici // 4. Créer le corps de la carte // ...votre code ici // 5. Assembler le tout // ...votre code ici return article; }
```

Questions à vous poser :

- Comment formater le titre ? (`${car.year} ${car.brand} ${car.model}`)
- Quel champ utiliser pour la description ?
- Comment construire l'URL avec l'ID ?

3.2 - Afficher toutes les voitures sur la page d'accueil

Objectif : Utiliser la fonction précédente pour afficher toutes les voitures récupérées de l'API.

Étapes :

1. **Sélectionner le conteneur** :
   - Utilisez `document.querySelector()` pour cibler la section avec la classe `.card-cont`
   - Stockez cette référence dans une variable
2. **Nettoyer le conteneur** :
   - IMPORTANT : Supprimez la carte exemple (Ferrari) du HTML
   - Ou videz le conteneur avec `innerHTML = ''` avant d'ajouter vos éléments
3. **Créer une fonction d'affichage** :

Pseudo-code :

JavaScript

```
FONCTION displayCars(cars): obtenir le conteneur vider le conteneur SI cars est vide OU null: afficher un message "Aucune voiture disponible" retourner POUR chaque voiture DANS cars: créer une carte avec createCarCard(voiture) ajouter la carte au conteneur
```

Orchestrer le chargement :

- Créer une fonction `init()` ou `loadHomePage()`
- Qui appelle `fetchAllCars()` puis `displayCars()`

Structure suggérée :

```javascript
async function init() { // 1. (Optionnel) Afficher un état de chargement // 2. Récupérer les données const cars = await fetchAllCars(); // 3. Afficher les données displayCars(cars); // 4. (Optionnel) Cacher l'état de chargement } // Lancer l'initialisation au chargement de la page init();
```

3.3 - Afficher les détails d'une voiture sur car.html

Objectif : Sur la page `car.html`, afficher les informations complètes d'une voiture spécifique.

Défis spécifiques :

1. **Récupérer l'ID depuis l'URL :**

```javascript
const urlParams = new
URLSearchParams(window.location.searc
const carId = urlParams.get('id');
```

- **Gérer les cas limites** :
  - Que faire si l'ID n'est pas présent dans l'URL ?
  - Que faire si l'ID n'existe pas dans l'API ?
  - Rediriger vers l'accueil ? Afficher un message d'erreur ?
- **Remplir les éléments HTML** :
  - Le titre principal (`<h2>`)
  - L'image principale
  - Le tableau de spécifications (année, marque, modèle, couleur, kilométrage, description, prix)

Structure du tableau à remplir :

Analysez le HTML existant dans `car.html`.
Vous avez un `<tbody>` avec plusieurs `<tr>`.

Approches possibles :

Approche 1 : Sélectionner chaque élément individuellement

```javascript
document.querySelector('h2').textCont
= `${car.year} ${car.brand}
${car.model}`;
document.querySelector('img').src =
car.imageUrl; // etc.
```

Approche 2 : Recréer le tableau dynamiquement

- Vider le tbody existant
- Créer les lignes une par une avec createElement

Approche 3 : Template HTML

- Garder le HTML comme template
- Remplir les valeurs avec JavaScript

Réflexions :

- Quelle approche est la plus maintenable ?
- Comment gérer le formatage du prix (ajouter des espaces, le symbole €) ?
- Comment gérer le kilométrage (formatage avec espaces) ?

Exemple de fonction d'initialisation pour car.html :

```javascript
async function initCarPage() { // 1. Récupérer l'ID const carId = urlParams.get('id'); // 2. Vérifier la présence de l'ID if (!carId) { // Gérer l'absence d'ID return; } // 3. Récupérer les données const car = await fetchCarById(carId); // 4. Vérifier que la voiture existe if (!car) { // Gérer la voiture inexistante return; } // 5. Afficher les données displayCarDetails(car); }
```

3.4 - Améliorer l'expérience utilisateur

États de chargement :

Pendant que les données sont récupérées, l'utilisateur doit savoir que quelque chose se passe.

Options :

1. Afficher un spinner Bootstrap
2. Afficher un texte "Chargement..."
3. Utiliser un skeleton screen (contours grisés des cartes)

Exemple de spinner Bootstrap :

```javascript
function showLoading(container) {
container.innerHTML = ` <div
class="text-center my-5"> <div
class="spinner-border"
role="status"> <span
class="visually-
hidden">Chargement...</span> </div>
</div> `; }
```

Gestion des erreurs côté utilisateur :

Si l'API ne répond pas ou retourne une erreur, l'utilisateur doit être informé.

Options :

1. Afficher un message d'erreur dans la page
2. Utiliser une alert Bootstrap
3. Rediriger vers une page d'erreur

Exemple d'alerte Bootstrap :

```javascript
function showError(container,
message) { const alert =
document.createElement('div');
alert.className = 'alert alert-
danger'; alert.textContent =
message;
container.appendChild(alert); }
```

💡 Points de vigilance

Performance

- **Évitez les reflows multiples** : Créez tous vos éléments en mémoire, puis ajoutez-les au DOM en une seule fois (ou utilisez un DocumentFragment)
- **Exemple** :

```javascript
const fragment =
document.createDocumentFragment();
cars.forEach(car => {
fragment.appendChild(createCarCard(ca
}); container.appendChild(fragment);
```

Accessibilité

- **Images** : Toujours renseigner l'attribut `alt`
- **Liens** : Avoir un texte descriptif ("See more" + titre de la voiture)
- **Contraste** : Vérifier la lisibilité du texte

Sécurité

- **Échappement des données** : Même avec createElement, attention aux données potentiellement malveillantes
- **Validation** : Vérifier que les données reçues de l'API sont au format attendu