

Découvrez les différents bugs du DOM

9–11 minutes

Dans la partie précédente, nous avons vu ensemble comment détecter et résoudre des bugs d'intégration. Dans cette partie, nous allons nous attaquer au JavaScript et aux bugs de DOM.

Découvrez les bugs liés à la sélection de noeuds

Initiez-vous aux bugs de sélection les plus courants

Imaginez-vous en train de coder votre application : vous avez ajouté le HTML et le CSS, les fichiers JavaScript sont bien reliés à votre page avec la balise `script` et vous souhaitez écouter un événement, par exemple, un clic de souris sur un bouton.

Et là, patatra ! Ça ne marche pas ! Vous avez soit une **erreur de référence**, autrement dit l'élément sélectionné n'existe pas, soit pire encore une **erreur silencieuse**.

Elles peuvent arriver via le DOM mais aussi via des API (et nous verrons ensemble comment les éviter).

Le type des objets et les méthodes `querySelector` et `querySelectorAll`

Ce qui m'amène sur un autre bug très fréquent : la mauvaise compréhension des méthodes `querySelector` et `querySelectorAll`.

- `querySelector` ne vous retourne toujours qu'**UN seul élément** même si la page en comporte plusieurs similaires. Par exemple, si vous sélectionnez une classe, l'élément retourné sera toujours la

première occurrence à votre sélection.

- `querySelectorAll` vous retourne toujours une **NodeList** (autrement dit, une liste de noeuds) HTML **MÊME** si votre page n'en contient qu'un. Autrement dit, vous aurez d'abord besoin d'extraire le, ou les, noeud(s) de votre tableau avant de pouvoir utiliser une méthode spécifique.

```
2
3 | const $userEmailInput = document.querySelectorAll('#user-email')
4 | const $userEmailErrorMsg = document.querySelector('.user-email-error-msg')
5
```

❗ Uncaught TypeError: \$userEmailInput.value is undefined signIn.js:13:30

checkUserEmailInput ...0.0.1:5500/js/pages/signIn.js:13
isFormValid ...0.0.1:5500/js/pages/signIn.js:36
<anonymous> ...0.0.1:5500/js/pages/signIn.js:41
EventListener.handleEvent* ...0.0.1:5500/js/pages/signIn.js:38
[\[En savoir plus\]](#)

checkUserEmailInput http://127.0.0.1:5500/js/pages/signIn.js:13
isFormValid http://127.0.0.1:5500/js/pages/signIn.js:36
<anonyme> http://127.0.0.1:5500/js/pages/signIn.js:41
(Asynchrone : EventListener.handleEvent)
<anonyme> http://127.0.0.1:5500/js/pages/signIn.js:38

Message d'erreur dans la console

Dans la capture d'écran ci-dessus, j'ai modifié la ligne 3 de `querySelector` à `querySelectorAll`. J'ai une erreur de type dans ma console. Cette dernière me dit que la propriété `value` n'est pas définie sur la variable `$userEmailInput`.


Quel est le type de variables qu'on manipule ?

Découvrez les bugs liés à la création de noeuds


Lors du développement d'une application, vous allez parfois vouloir ajouter des **éléments dynamiques** sur votre site en passant, par exemple, par une **API**. Cette dernière va vous permettre de récupérer des données, créer une liste avec ces données et les afficher.

Vos capteurs


Filter par status : Tous




Capteur #1
Localisation : Rue Georges Lardennois
Status : actif



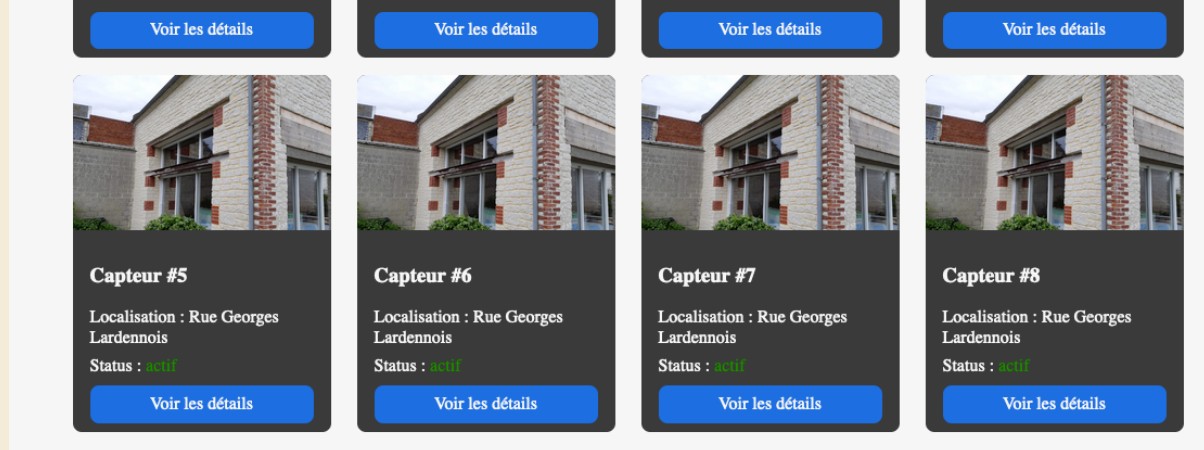
Capteur #2
Localisation : Rue Georges Lardennois
Status : inactive



Capteur #3
Localisation : Rue Georges Lardennois
Status : actif



Capteur #4
Localisation : Rue Georges Lardennois
Status : actif



La page d'accueil des façades et leurs capteurs sur Façadia

Dans le cadre de Façadia, je crée chacune des fiches de capteur de façon dynamique et je les injecte après dans le projet.

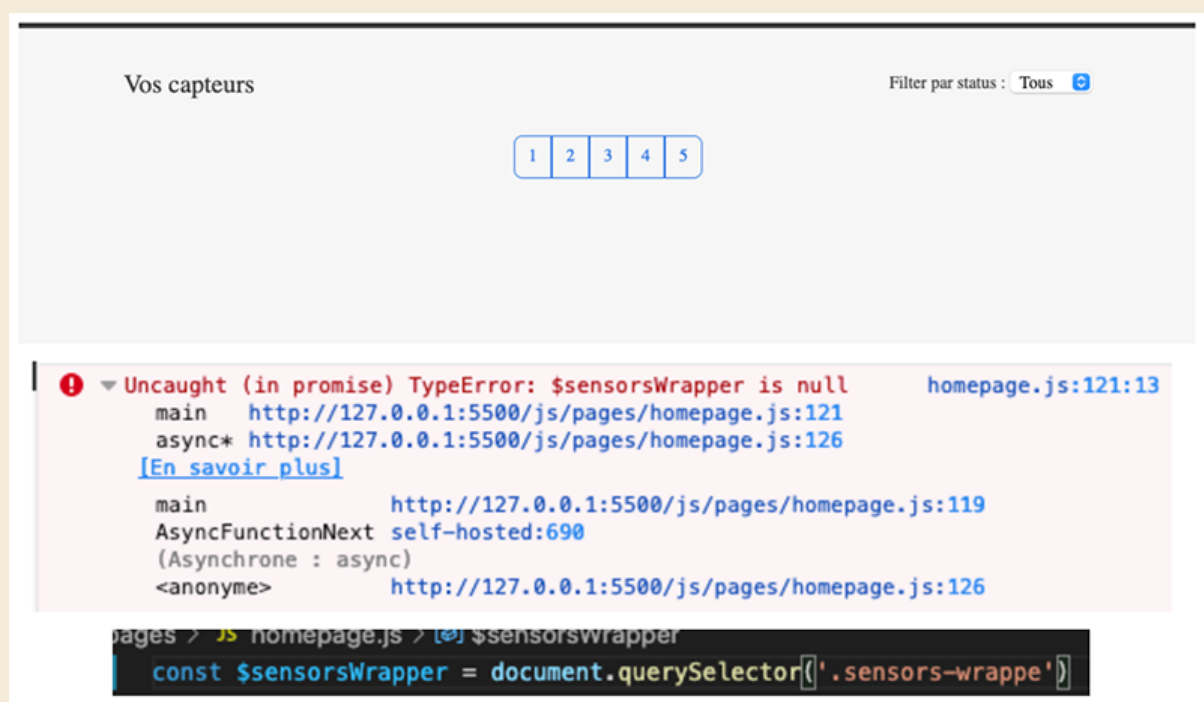
Initiez-vous aux bugs de création les plus courants

Les bugs de création arrivent le plus souvent parce que vous avez mal créé ou mal ajouté un élément sur votre DOM. Il peut aussi parfois arriver que vous ayez un bug parce que vous avez mal sélectionné un élément (ce dernier n'existe pas) et vous ne pouvez donc pas l'ajouter



.

Ça ne vous rappelle rien ?



Le bug sur le site de Façadia où les capteurs ne s'affiche pas, le message d'erreur associé et le code concerné

Dans l'exemple ci-dessus, j'ai mal sélectionné l'élément

`.sensors-wrapper` en oubliant le "R" de `.sensors-wrapper`.

Au moment d'ajouter chacune des `div` créées, j'ai une erreur de type : la variable `$sensorsWrapper` est alors nulle.

Le mieux à faire dans ce type de cas est de bien réaliser la création de votre noeud :

- utilisez `createElement` pour créer votre noeud ;
- puis, servez-vous de `textContent` pour lui ajouter un contenu textuel (si c'est possible)
- puis, tirez parti de l'objet `classList` pour ajouter, lister ou enlever une classe.
- et enfin, ajoutez le noeud nouvellement créé à son nœud "parent" via `appendChild`.

Le plus souvent, c'est quand vous essayez de tout faire à la fois que vous risquez de vous prendre les pieds dans le tapis. Je sais que cela peut sembler contre-intuitif : on a souvent tendance à dire que les développeurs sont feignants. Cela dit, les développeurs aiment aussi le "beau" code, celui qui est robuste et compréhensible pour tous



.

La méthode `appendChild` et ses messages d'erreur

Dans le cadre de la méthode `appendChild`, vous serez susceptible d'avoir deux grands types de messages d'erreur :

- la `TypeError` (ou erreur de type) (que vous avez vue un peu plus haut). Cette erreur sera le plus souvent présente quand vous aurez mal sélectionné un élément sur le DOM.
- la `ReferenceError` (ou erreur de référence) qui correspondra à une variable non définie. Dans l'exemple ci-dessous, je supprime délibérément une lettre (la lettre `l` de `title`) sur ma variable `sensorInfoTitle`.

```
❗ Uncaught (in promise) ReferenceError: $sensorInfoTite is not defined homepage.js:48:5
  createSensorCardInfo ...127.0.0.1:5500/js/pages/homepage.js:48
  createSensorCard ...127.0.0.1:5500/js/pages/homepage.js:62
  main ...27.0.0.1:5500/js/pages/homepage.js:121
  async* ...27.0.0.1:5500/js/pages/homepage.js:126
[En savoir plus]
  main http://127.0.0.1:5500/js/pages/homepage.js:119
  AsyncFunctionNext self-hosted:690
  (Asynchrone : async)
  <anonyme> http://127.0.0.1:5500/js/pages/homepage.js:126
```

Le message d'erreur suite à la faute d'orthographe

Découvrez les bugs liés à la modification de noeuds existants

Côté bug et modification des nœuds existants, le bug le plus courant est celui où vous essayez d'utiliser une propriété ou une méthode qui n'existe pas sur le nœud sélectionné. Une fois encore, il est important de connaître le type de variable que vous manipulez et de comprendre le message d'erreur associé



.

Utilisez le CSS pour ajouter du style à vos éléments

Il est important que nous revenions sur un sujet particulier : le CSS et le DOM. Admettons que vous soyez en train de travailler sur un formulaire de contact et que vous souhaitiez gérer les cas où le champ de votre formulaire est correct et incorrect. Comment le feriez-vous ?

Il existe deux solutions :

- vous utilisez le JavaScript pour ajouter et supprimer des classes, par exemple ; chaque classe étant liée à un sélecteur CSS et se chargeant d'ajouter le style.
- OU vous ajoutez directement du style via le JavaScript sur le nœud HTML avec la propriété `.style` .

```
const checkUserPasswordInput = () => {
  const isUserPasswordValid =
$userPasswordInput.value === USER_PASSWORD
  if (isUserPasswordValid) {
```

```

        // J'ajoute une classe HTML "hidden" à mon
noeud userPasswordErrorMsg, elle a pour règle css un
display: none

        $userPasswordErrorMsg.classList.add('hidden')
    } else {

        // J'enlève une classe HTML "hidden" à mon
noeud userPasswordErrorMsg, elle a pour règle css un
display: none

$userPasswordErrorMsg.classList.remove('hidden')
    }

    return isUserPasswordValid
}

```

Dans le code ci-dessus, vous avez la première solution. Ici, j'ajoute (ou supprime une classe) en fonction de la condition (si le mot de passe de l'utilisateur est valide).

Voici la deuxième solution ; on injecte directement du style depuis le JavaScript dans le HTML :

```

const checkUserEmailInput = () => {

    const isUserEmailValid =
$userEmailInput.value.toLowerCase() === USER_EMAIL

    if (isUserEmailValid) {

        $userEmailErrorMsg.style.display = "none"
    } else {

        $userEmailErrorMsg.style.display = "block"
    }

    return isUserEmailValid
}

```

Personnellement, j'ai tendance à utiliser la première solution. Mais pourquoi ?

- D'une part, parce qu'il y a une **violation des responsabilités** : le HTML est là pour le contenu, le CSS est là pour la mise en page et le JavaScript pour gérer les interactions.
- D'autre part, parce que vous multipliez les sources "de vérité", à savoir là où les règles s'appliquent. C'est une source de bug très très commune



.

Euhhhh c'est quoi la violation des responsabilités ?

Bonne question !

Cela vient du principe de responsabilité unique qui dit que chaque fonction ou module ne devrait avoir qu'une responsabilité. Dans le cadre du trio HTML, CSS et JavaScript : le HTML a la responsabilité du contenu, le CSS de la mise en page et le JavaScript des interactions.

Dans ce type, tirez parti de [l'excellent objet classList](#) et de ses méthodes `.toggle()`, `.add()`, `.remove()`, etc.

Découvrez les bugs liés à la suppression de noeuds

Dernier focus avant de passer à la pratique : la **suppression de noeuds**.

Vous est-il parfois arrivé d'avoir envie de masquer un nœud en le supprimant plutôt qu'en le masquant côté CSS ? Le plus souvent, les bugs liés à la suppression de noeuds viendront du fait qu'on a **supprimé** le nœud du DOM au lieu de simplement le masquer ou alors d'une mauvaise utilisation de la méthode `removeChild()`

En effet, la méthode `removeChild()` permet de supprimer un nœud enfant contenu dans un nœud parent. Cela dit, si vous ne sélectionnez pas directement le noeud parent mais un autre noeud, vous pouvez avoir une erreur comme ci-dessous :

Uncaught DOMException: Node.removeChild: The node to be removed is `signIn` is:15


```
Uncaught DOMException: Node.removeChild: The node to be removed is not a child of this node  
checkUserEmailInput http://127.0.0.1:5500/js/pages/signIn.js:15  
isFormValid http://127.0.0.1:5500/js/pages/signIn.js:38  
<anonyme> http://127.0.0.1:5500/js/pages/signIn.js:43  
(Asynchrone : EventListener.handleEvent)  
<anonyme> http://127.0.0.1:5500/js/pages/signIn.js:40
```

Le message d'erreur

À vous de jouer !



C'est l'heure de la pratique !

Avant de regarder la vidéo ci-dessous, mettez-vous sur la branche [partie-3/chapitre-1/section-5](#) et essayez de traquer les bugs de DOM sur notre projet Façadia. J'ai créé une issue par bug sur le repository Github. Nous nous intéresserons dans le prochain chapitre à la résolution de ces bugs.

- L'issue de création : <https://github.com/tdimnet/debuggez-l-interface-de-votre-site/issues/2>
- L'issue de modification : <https://github.com/tdimnet/debuggez-l-interface-de-votre-site/issues/3>

En résumé

- Les bugs de sélection d'éléments sont souvent dus à des éléments mal sélectionnés. Cela peut parfois vous donner des erreurs silencieuses ou des erreurs de référence ou de type.
- Attention à ne pas trop recourir à la méthode `insertAdjacentHTML` qui peut parfois être moins lisible et plus source de bugs que la méthode `createElement`.
- Si vous devez modifier le style d'un nœud, essayez le plus souvent de passer par l'objet `classList` et ses méthodes `add`, `remove`

ou `toggle` . Cela vous permettra d'éviter des bugs liés à la modification de votre DOM.

- Lors de la suppression d'un nœud, posez-vous toujours la question de le masquer via le CSS ou de le supprimer définitivement du DOM. Dans le cadre d'un formulaire de contact, il peut être plus intéressant de le masquer.

Maintenant que vous connaissez les sources de bug possibles sur le DOM, vous allez pouvoir apprendre à utiliser des outils pour les déboguer plus facilement. 😊

- <#>