5 minutes

Mise en place du JavaScript pour le formulaire

Modification de `form.js`

Nous allons commencer par la gestion du formulaire en `JavaScript`.

Voici le code que nous allons expliquer :

```
const form = document.querySelector("form");
const errorElement = document.querySelector("#errors");
let errors = [];

form.addEventListener("submit", async event => {
  event.preventDefault();
  const formData = new FormData(form);
  const article = Object.fromEntries(formData.entries());
  if (formIsValid(article)) {
    const json = JSON.stringify(article);
    // Nous ferons la requête ici !
  }
});

const formIsValid = article => {
  if (
    !article.author ||
    !article.category ||
    !article.content
  ) {
    errors.push("Vous devez renseigner tous les champs");
  } else {
    errors = [];
  }
  if (errors.length) {
    let errorHTML = "";
    errors.forEach(e => {
      errorHTML += `<li>${e}</li>`;
    });
    errorElement.innerHTML = errorHTML;
    return false;
```

```
  } else {
    errorElement.innerHTML = "";
    return true;
  }
};
```

Nous commençons par créer une référence pour notre formulaire et une pour une liste non ordonnées que nous allons créer pour afficher nos erreurs.

Ensuite, nous ajoutons un écouteur sur l'événement `submit` du formulaire.

Nous empêchons le comportement par défaut qui est de recharger la page sur le `submit` avec `event.preventDefault()`.

Nous utilisons l'objet `FormData` pour parser tous les champs du formulaire.

Ensuite, nous utilisons la méthode `Object.fromEntries()` qui permet de transformer un itérable de paires de clé / valeur en un objet `JavaScript`.

Nous lui passons `formData.entries()` qui retourne justement un itérable de paires de clé / valeur.

Nous avons donc un objet `JavaScript` en sorti avec des paires clé / valeur facilement convertible en `JSON`.

Enfin, nous mettons en place une fonction de validation pour les champs du formulaire qui va vérifier des conditions avant de convertir l'objet en `JSON` et de l'envoyer au serveur.

Dans cette fonction `formIsValid`, nous allons vérifier que tous les champs sont remplis. Si ce n'est pas le cas, nous allons ajouter une erreur à notre tableau d'erreurs.

Si ce tableau contient au moins une erreur nous allons toutes les afficher dans des éléments de liste.

Cette manière est la bonne car vous pouvez ajouter autant d'erreurs et de validations que vous souhaitez. Vous pourriez par exemple faire :

```
errors = [];
if (!article.author || !article.category || !article.content) {
  errors.push("Vous devez renseigner tous les champs");
}
if (article.content.length < 20) {
  errors.push("Le contenu de votre article est trop court !");
}
```

Vous pouvez mettre ainsi des dizaines de validation sans problème et les afficher.

Enfin, la fonction retourne un booléen indiquant si le formulaire est valide ou non. Si il y a au moins une erreur, le formulaire n'est pas valide.

Modification de `form.html`

Nous ajoutons une liste non ordonnée pour nos erreurs, juste au dessus du `form-btn-container` :

```
<ul class="text-error" id="errors"></ul>
```

Modification du `partial _classes.scss`

Nous ajoutons une classe pour colorer les erreurs que nous affichons :

```
.text {
  &-error {
    color: var(--error);
  }
}
```

Modification du `partial _variables.scss`

Nous n'avons plus qu'à ajouter la variable correspondante :

```
  --error: #e74c3c;
```

Code exécutable

Vous pouvez voir directement où nous en sommes.

A noter que l'éditeur en ligne utilise `Parcel` et pas `Webpack` donc vous ne pourrez pas voir la configuration de `Webpack`, c'est normal.

A noter également que pour la même raison, les liens ne sont pas les mêmes ce qui est normal car nous n'utilisons pas `Webpack` dans l'éditeur en ligne :
https://codesandbox.io/embed/js-c16-l8-qwfdc