Valider les formulaires

La validation d'un formulaire du côté front-end permet de vérifier que les données saisies par l'utilisateur sont correctes avant de les envoyer au serveur. Cela permet d'éviter des erreurs et des pertes de temps en cas de saisie incorrecte. On indique à l'utilisateur les erreurs de saisie pour qu'il puisse les corriger.

Les champs de formulaire possèdent des attributs qui permettent de définir des contraintes de saisie. En ajoutant ces attributs, le navigateur va valider automatiquement les champs en fonction des contraintes définies. Si un champ ne respecte pas les contraintes, le navigateur affiche un message d'erreur et empêche l'envoi du formulaire.

Attributs de contrainte des champs

- `required` : Champ obligatoire. L'utilisateur doit remplir le champ pour que le formulaire soit valide. Le champ ne doit pas être vide.
- `minlength` : Longueur minimale du champ. Pour les champs de type texte, le nombre de caractères doit être supérieur ou égal à la valeur de l'attribut. Fonctionne aussi avec les textarea.
- `maxlength` : Longueur maximale du champ. Pour les champs de type texte, le nombre de caractères doit être inférieur ou égal à la valeur de l'attribut. Fonctionne aussi avec les textarea.
- `min` : Valeur minimale du champ. Pour les champs de type number, la valeur doit être supérieure ou égale à la valeur de l'attribut. Fonctionne aussi avec les champs de type date et range.
- `max` : Valeur maximale du champ. Pour les champs de type number, la valeur doit être inférieure ou égale à la valeur de l'attribut. Fonctionne aussi avec les champs de type date et range.
- `type` : Certains types de champs possèdent sont validés automatiquement. Les types `email`, `url`et `number` sont validés automatiquement par le navigateur.
- `pattern` : Expression régulière qui doit correspondre à la valeur du champ. Permet de définir des contraintes de saisie plus complexes. Nous verrons les expressions régulières au cours 13.
- `accept` : Permet de définir les types de fichiers acceptés dans un champ de type file. Ex: `accept="image/png"` pour les images.

Validation manuelle des formulaires

Blocage de la soumission du formulaire

Lors d'une validation manuelle, il est essentiel de bloquer la soumission du formulaire avant de procéder à la validation. Pour cela, on utilise l'événement `submit` du formulaire. On empêche l'envoi du formulaire en utilisant la méthode `preventDefault()` de l'objet `event`.

```
const form = document.querySelector("form");

form.addEventListener("submit", function (event) {
    event.preventDefault(); // Bloque l'envoi du formulaire
    // Validation du formulaire
    // Si le formulaire est valide, on peut envoyer les données
    if (form.checkValidity()) {
        form.submit();
    }
});
```

Validation des champs

Pour valider les champs, on utilise la méthode `checkValidity()` des champs de formulaire ou sur le formulaire lui-même. Cette méthode renvoie `true` si le champ est valide et `false` sinon.

Si un champ est invalide, un événement `invalid` est déclenché sur le champ. On peut écouter cet événement pour afficher un message d'erreur à l'utilisateur.

On peut également utiliser la propriété `validity` pour obtenir des informations sur les erreurs de validation sur un champ précis. Cette propriété n'est pas accessible sur le formulaire.

La propriété `validationMessage` permet d'obtenir le message d'erreur associé à un champ invalide. Ce message est généré automatiquement par le navigateur en fonction des contraintes de saisie définies.

```
const champ = document.querySelector("input[type='text']");
const formulaire = document.querySelector("form");

const formulaireEstValide = formulaire.checkValidity(); // Vérifie si le formulaire
est valide
const estValide = champ.checkValidity(); // Vérifie si le champ est valide

champ.addEventListener("change", function () {
    champs.querySelector("span").textContent = "";
    champ.checkValidity();
});

champ.addEventListener("invalid", function () {
    const messageErreur = champ.validationMessage;
    champs.querySelector("span").textContent = messageErreur;
});
```

Validation personnalisée

Il est possible de définir des contraintes de validation personnalisées en utilisant la méthode `setCustomValidity()` des champs de formulaire. Cette méthode permet de définir un message d'erreur personnalisé pour un champ invalide.

Pour rendre un champ de nouveau valide, il faut appeler la méthode `setCustomValidity("")` en lui passant une chaîne vide. Sinon, le champ restera invalide.

```
const champ = document.querySelector("input[type='text']");
champ.setCustomValidity("Message d'erreur personnalisé");

champ.addEventListener("change", function () {
    champ.setCustomValidity(""); // Champ valide
});
```

Exemple de validation d'un formulaire

```
<form>
    <label for="nom">Nom :</label>
    <input type="text" id="nom" name="nom" required minlength="2" maxlength="50" />
    <span class="message-erreur"></span>

    <label for="courriel">courriel :</label>
    <input type="email" id="courriel" name="courriel" required />
    <span class="message-erreur"></span>

    <label for="mdp">Mot de passe :</label>
    <input type="password" id="mdp" name="mdp" required minlength="8" />
    <span class="message-erreur"></span>

    <button type="submit">Envoyer</button>
</form>
```

```
const form = document.querySelector("form");
const champs = document.querySelectorAll("input");
const boutonSubmit = document.querySelector("input[type='submit']");
let resume = {};

function init() {
    champs.forEach(function (champ) {
        champ.addEventListener("change", onChangementChamps);
        champ.addEventListener("invalid", onInvalid);
    });

    form.addEventListener("submit", function (event) {});
```

```
}

function onSubmit(evenement) {
    evenement.preventDefault(); // Bloque l'envoi du formulaire

    if (form.checkValidity()) {
        console.log(resume); //Envoie les données de façon asynchrone ou synchrone
        form.submit(); //synchrone
        form.reset(); //Réinitialise le formulaire
    }
}

function onChangementChamps(evenement) {
    const declencheur = evenement.currentTarget;
    const name = declencheur.name;
    const value = declencheur.value;

    //Validation personnalisée
    if (declencheur.type === "email") {
        if (!value.endsWith("cmaisonneuve.qc.ca")) {
            declencheur.setCustomValidity("Le courriel doit se terminer par
cmaisonneuve.qc.ca");
        } else {
            declencheur.setCustomValidity("");
        }
    }

    if (declencheur.checkValidity() == true) {
        declencheur.nextElementSibling.textContent = "";
        resume[name] = value;
    }

    verifierFormulaireValide();
}

function onInvalid(evenement) {
    const declencheur = evenement.currentTarget;
    const messageErreur = declencheur.validationMessage;
    declencheur.nextElementSibling.textContent = messageErreur;
}
function verifierFormulaireValide() {
    if (form.checkValidity()) {
        boutonSubmit.disabled = false;
    } else {
        boutonSubmit.disabled = true;
    }
}

init();
```

Rétroaction visuelle via CSS

Il est possible de donner une rétroaction visuelle à l'utilisateur en utilisant les pseudo-classes CSS `:valid` et `:invalid`. Ces pseudo-classes permettent de cibler les champs valides et invalides.

```
input:valid {
    border-color: green;
}

input:invalid {
    border-color: red;
}

input:invalid + span {
    display: block;
    color: red;
}

input:valid + span {
    display: none;
}
```