7 minutes

Création d'un formulaire et introduction à useForm()

Le `hook useForm()`

Le `hook useForm()` est le hook principal de la librairie. Il permet de gérer un formulaire.

Voici les options possibles que nous étudierons au fur et à mesure :

```
useForm({
  mode: 'onSubmit',
  reValidateMode: 'onChange',
  defaultValues: {},
  resolver: undefined,
  context: undefined,
  criteriaMode: "firstError",
  shouldFocusError: true,
  shouldUnregister: false,
  shouldUseNativeValidation: false,
  delayError: undefined
})
```

Copier

Le `hook` retourne également un ensemble de méthodes et d'objets permettant de contrôler finement le formulaire : `register`, `unregister`, `formState`, `watch`, `handleSubmit`, `reset`, `resetField`, `setError`, `clearErrors`, `setValue`, `setFocus`, `getValues`, `getFieldState`, `trigger`, et `control`.

Nous les verrons au fur et à mesure de nos besoins.

Voici un premier résumé des méthodes principales que nous approfondirons ensuite.

La méthode `handleSubmit()`

Cette méthode permet de prévenir le comportement de la soumission du formulaire, c'est-à-dire le rafraîchissement de la page.

Elle peut prendre deux fonctions de rappel en argument :

- Le premier argument est **une fonction de rappel qui est invoquée lorsque le formulaire est envoyé** (*lorsque le bouton de `type submit`, qui est le type par défaut des boutons dans les formulaires, est cliqué*). **Cette fonction de rappel reçoit en arguments les données du formulaire et l'objet d'événement**.
- Le deuxième argument est une **fonction de rappel qui est invoquée lorsqu'une erreur est émise** lors de l'envoi du formulaire. **Cette fonction de rappel reçoit en arguments les erreurs du formulaire et l'objet d'événement**.

Voici un exemple simple, nous verrons à quoi sert `register()` juste après :

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, handleSubmit } = useForm();
  const onSubmit = (data, e) => console.log(data, e);
  const onError = (errors, e) => console.log(errors, e);

  return (
```

```
    <form onSubmit={handleSubmit(onSubmit, onError)}>
      <input {...register("firstName")} />
      <input {...register("lastName")} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Copier

La méthode `register()`

Cette méthode permet d'enregistrer un champ et d'y appliquer des règles de validation.

Lorsqu'un champ est enregistré sa valeur sera disponible lors de l'envoi (*dans l'objet `data` de la première fonction de rappel passé à `handleSubmit()`*) et lors de la validation.

Il faut obligatoirement fournir une clé comme nom du champ à enregistrer. Ce nom doit être unique pour le formulaire.

Elle fonctionne de cette manière :

```
const { onChange, onBlur, name, ref } = register('prenom');

<input
  onChange={onChange}
  onBlur={onBlur}
  name={name}
  ref={ref}
/>
```

Copier

Mais nous pouvons utiliser la syntaxe raccourcie :

```
<input {...register('prenom')} />
```

Copier

Cette méthode peut prendre un grand nombre d'options de validation que nous étudierons.

La méthode `getValues()`

Cette méthode permet de lire les valeurs du formulaire sans re-déclencher un rendu ni s'abonner aux changements des champs avec des écouteurs d'événement.

Prenons un exemple avec deux champs et supposons que nous ayons entré `a` dans le champ `test` et `b` dans le champ `test1` :

```
import { useForm } from "react-hook-form";

export default function App() {
  const { register, getValues } = useForm();

  return (
    <form>
      <input {...register("test")} />
      <input {...register("test1")} />
```

```
      <button
        type="button"
        onClick={() => {
          const valeurs = getValues(); // { test: "a", test1: "b" }
          const valeurDunChamp = getValues("test"); // "a"
          const valeurDePlusieursChamps = getValues(["test", "test1"]); // ["a", "b"]
        }}
      >
        Get Values
      </button>
    </form>
  );
}
```

Copier

La méthode `watch()`

Cette méthode permet de surveiller un ou plusieurs champs et de retourner leurs valeurs. C'est utile pour afficher à un autre endroit la valeur d'un champ.

Attention ! Contrairement à `getValues()`, la méthode `watch()` déclenche de nombreux rendus supplémentaires, à chaque changement du ou des champs surveillés.

Si vous ne passez rien en argument, tous les champs seront surveillés.