# 3 minutes

# Gestion de la soumission du formulaire

## Réinitialisation du formulaire

**`useForm()` met à disposition la méthode `reset()` permettant de réinitialiser le formulaire.**

Si vous utilisez la méthode sans argument, le formulaire sera réinitialisé avec les valeurs initiales.

Vous pouvez passer en premier argument les valeurs des champs après la réinitialisation et dans ce cas il est recommandé de toutes les fournir.

Vous pouvez réinitialiser le formulaire dans un effet après que le formulaire ait été bien envoyé :

```
useEffect(() => {
  reset({
    data: 'test'
  })
}, [isSubmitSuccessful])
```

Copier

## L'état du formulaire

**`useForm()` met à disposition un objet `formState` contenant un grand nombre de variables d'état relatives à l'état du formulaire :**

- **`isDirty`** : passe à `true` dès lors que l'utilisateur modifie un seul champ.
- **`dirtyFields`** : objet contenant les champs que l'utilisateur a modifié.
- **`touchedFields`** : objet contenant les champs que l'utilisateur a touché.
- **`isSubmitted`** : booléen qui vaut `true` dès lors que le formulaire a été envoyé. Reste à `true` tant que le formulaire n'est pas réinitialisé avec `reset()`.
- **`isSubmitSuccessful`** : booléen qui vaut `true` si le formulaire a été envoyé sans échec (sans rejet de promesse ou sans erreur ayant été renvoyée dans la fonction de rappel `handleSubmit()`).
- **`isSubmitting`** : booléen qui vaut `true` si le formulaire est en cours d'envoi.
- **`submitCount`** : nombre de fois que le formulaire a été envoyé.
- **`isValid`** : booléen qui vaut `true` si le formulaire n'a pas d'erreur (**il faut que le mode du formulaire soit `onChange`, `onBlur` ou `onTouched`**).
- **`isValidating`** : booléen qui vaut `true` si le formulaire est en cours de validation (validation asynchrone).
- **`errors`** : objet contenant les erreurs du formulaire.

Grâce à l'état du formulaire, vous pouvez par exemple désactiver le bouton d'envoi :

```
const { isDirty, isValid, isSubmitting } = formState;

<button disabled={isDirty || isValid || !isSubmitting} />;
```

Copier

## Définir manuellement des erreurs

**`useForm()` met à disposition la méthode `setError()` permettant de définir manuellement des erreurs sur le formulaire.**

**Le plus souvent cette méthode est utilisée dans la méthode `handleSubmit()` pour définir une erreur lors d'une validation asynchrone** (par exemple une API qui retourne une erreur de validation).

Le premier argument de la méthode est le nom du champ concerné par l'erreur, le second argument est un objet qui contient une clé `type` et une clé `message`.

Par exemple :

```
setError('nomChamp', { type: 'unType', message: 'Un message d\'erreur' });
```

Copier

**A noter que si le champ n'est pas enregistré avec `register()`, l'erreur ne sera enlevée qu'avec la méthode `clearErrors()`.**

La méthode `clearErrors()` retire toutes les erreurs si aucun argument ne lui est passé. Il est possible de ne retirer que certaines erreurs en passant une chaine de caractères ou un tableau de chaines de caractères (par exemple `clearErrors(["firstName", "lastName"])`).