3 minutes

Validation avec Yup

Gestion de la validation avec `Yup`

`Yup` est un validateur de schémas d'objets `JavaScript`.

Cela a l'air compliqué, mais c'est très simple : cela consiste à définir un schéma comme "modèle" pour vos objets et de vérifier qu'ils respectent ces schémas.

`Yup` et `React Hook Form`

Pour utiliser `Yup` avec `React Hook Form` il faut utiliser le `resolver` prévu par la librairie, que nous avons déjà installé :

```
import { yupResolver } from '@hookform/resolvers/yup';
// ...
const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    resolver: yupResolver(schema),
  });
```

Copier

Définition des schémas

La librairie `Yup` est tout simplement géniale !

Vous pouvez tout faire : de la validation synchrone, asynchrone, des validations multi-champs, et utiliser des dizaines de validateurs pour chaque type.

Impossible de tout voir, et ce n'est pas intéressant. Nous allons voir les principaux validateurs :

- **`Yup.object()`** : permet de définir un schéma d'objet (il est possible de définir des schémas pour des tableaux, ou seulement des nombres, des chaînes de caractères etc). Mais pour les formulaires ce sera un objet.
- **`Yup.string()`** : permet de définir un schéma de chaîne de caractères. Notez que par défaut, `Yup` appellera `toString()` sur la valeur avant de déterminer si elle est valide.
- **`Yup.min(limit: number, message?: string)`** : permet de définir une longueur minimale pour la chaîne de caractères en premier argument. En deuxième argument vous pouvez définir optionnellement le message d'erreur qui sera retourné.
- **`Yup.required(message?: string)`** : permet de rendre obligatoire une propriété et de renvoyer optionnellement un message passé en paramètre.
- **`Yup.email(message?: string)`** : permet de valider une chaîne de caractères comme étant un email grâce à une expression régulière.
- **`Schema.typeError(message: string)`** : permet de définir un message d'erreur si le type ne correspond pas à celui du schéma.
- **`Schema.oneOf(tableau, message?: string)`** : permet d'accepter uniquement les valeurs du tableau passé en premier argument. Le deuxième argument permet de préciser un message d'erreur.

- **`Yup.ref(path: string)`** : permet de faire référence à un autre champ dont le chemin est précisé en premier argument.
- **`Schema.test(name: string, message: string, test: function)`** : permet de créer un test personnalisé. Le premier argument est le nom du validateur, le second le message d'erreur et le troisième la fonction de validation qui doit retourner un booléen.

Vous pouvez retrouver l'ensemble des validateurs ici.

Exemple de la vidéo

Voici le code de l'exemple de la vidéo :

```
import * as yup from 'yup';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';

function App() {
  const schema = yup.object({
    name: yup
      .string()
      .required('Le champ est obligatoire')
      .min(2, 'Trop court')
      .max(5, 'Trop long')
      .test('isYes', "Vous n'avez pas de chance", async () => {
        const response = await fetch('https://yesno.wtf/api');
        const result = await response.json();
        console.log(result);
        return result.answer === 'yes';
      }),
    age: yup
      .number()
      .typeError('Veuillez entre un nombre')
      .min(18, 'Trop jeune'),
    password: yup
      .string()
      .required('Le mot de passe est obligatoire')
      .min(5, 'Mot de passe trop court')
      .max(10, 'Mot de passe trop long'),
    confirmPassword: yup
      .string()
      .required('Vous devez confirmer votre mot de passe')
      .oneOf(
        [yup.ref('password'), ''],
        'Les mots de passe ne correspondent pas'
      ),
  });

  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm({
    defaultValues: {
      name: '',
    },
    resolver: yupResolver(schema),
  });

  function submit(values) {
    console.log(values);
  }

  return (
    <div
```

```
        className="d-flex justify-content-center align-items-center"
        style={{ backgroundColor: '#fefefe', height: '100vh', width: '100%' }}
      >
        <form onSubmit={handleSubmit(submit)}>
          <div className="d-flex flex-column mb-20">
            <label htmlFor="name" className="mb-5">
              Nom
            </label>
            <input id="name" type="text" {...register('name')} />
            {errors?.name && (
              <p style={{ color: 'red' }}>{errors.name.message}</p>
            )}
          </div>
          <div className="d-flex flex-column mb-20">
            <label htmlFor="name" className="mb-5">
              Age
            </label>
            <input
              id="age"
              type="number"
              {...register('age', {
                valueAsNumber: true,
              })}
            />
            {errors?.age && <p style={{ color: 'red' }}>{errors.age.message}</p>}
          </div>
          <div className="d-flex flex-column mb-20">
            <label htmlFor="password" className="mb-5">
              Mot de passe
            </label>
            <input id="password" type="password" {...register('password')} />
            {errors?.password && (
              <p style={{ color: 'red' }}>{errors.password.message}</p>
            )}
          </div>
          <div className="d-flex flex-column mb-20">
            <label htmlFor="confirmPassword" className="mb-5">
              Confirmation du mot de passe
            </label>
            <input
              id="confirmPassword"
              type="password"
              {...register('confirmPassword')}
            />
            {errors?.confirmPassword && (
              <p style={{ color: 'red' }}>{errors.confirmPassword.message}</p>
            )}
          </div>
          <button className="btn btn-primary">Sauvegarder</button>
        </form>
      </div>
    );
}

export default App;
```

Copier