Exercice 4 - Formulaire avec validation avancée

Objectif

Créer un formulaire d'inscription avec validation en temps réel.

Concepts abordés

- Formulaires contrôlés
- Validation synchrone et asynchrone
- Gestion des erreurs par champ
- État de soumission

Énoncé

Créer un formulaire `SignupForm` avec :

1. Champs : nom, email, mot de passe, confirmation
2. Validation en temps réel (au blur ou à la frappe)
3. Règles de validation :
   - Nom : minimum 2 caractères
   - Email : format valide
   - Mot de passe : minimum 8 caractères, 1 majuscule, 1 chiffre
   - Confirmation : doit correspondre au mot de passe
4. Bouton désactivé si erreurs
5. Affichage des erreurs sous chaque champ

Correction

```
import { useState } from 'react';
import './SignupForm.css';

// Règles de validation
const validators = {
  name: (value) => {
    if (!value.trim()) return 'Le nom est requis';
    if (value.trim().length < 2) return 'Minimum 2 caractères';
    return null;
  },

  email: (value) => {
    if (!value.trim()) return 'L\'email est requis';
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(value)) return 'Email invalide';
    return null;
  },
```

```
  password: (value) => {
    if (!value) return 'Le mot de passe est requis';
    if (value.length < 8) return 'Minimum 8 caractères';
    if (!/[A-Z]/.test(value)) return 'Au moins une majuscule';
    if (!/[0-9]/.test(value)) return 'Au moins un chiffre';
    return null;
  },

  confirmPassword: (value, formData) => {
    if (!value) return 'Confirmez le mot de passe';
    if (value !== formData.password) return 'Les mots de passe ne correspondent
pas';
    return null;
  }
};

function SignupForm() {
  // State du formulaire
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
  });

  // State des erreurs
  const [errors, setErrors] = useState({});

  // State des champs touchés (pour afficher erreurs au bon moment)
  const [touched, setTouched] = useState({});

  // State de soumission
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [submitSuccess, setSubmitSuccess] = useState(false);

  // Valider un champ
  function validateField(name, value) {
    const validator = validators[name];
    if (validator) {
      return validator(value, formData);
    }
    return null;
  }

  // Valider tout le formulaire
  function validateForm() {
    const newErrors = {};
    Object.keys(formData).forEach(field => {
      const error = validateField(field, formData[field]);
      if (error) newErrors[field] = error;
    });
    return newErrors;
  }
```

```
// Handler de changement
function handleChange(e) {
  const { name, value } = e.target;

  setFormData(prev => ({ ...prev, [name]: value }));

  // Valider si le champ a été touché
  if (touched[name]) {
    const error = validateField(name, value);
    setErrors(prev => ({ ...prev, [name]: error }));
  }

  // Re-valider confirmPassword si password change
  if (name === 'password' && touched.confirmPassword) {
    const confirmError = validators.confirmPassword(
      formData.confirmPassword,
      { ...formData, password: value }
    );
    setErrors(prev => ({ ...prev, confirmPassword: confirmError }));
  }
}

// Handler de blur (quand on quitte un champ)
function handleBlur(e) {
  const { name, value } = e.target;

  // Marquer comme touché
  setTouched(prev => ({ ...prev, [name]: true }));

  // Valider
  const error = validateField(name, value);
  setErrors(prev => ({ ...prev, [name]: error }));
}

// Handler de soumission
async function handleSubmit(e) {
  e.preventDefault();

  // Marquer tous les champs comme touchés
  const allTouched = Object.keys(formData).reduce(
    (acc, key) => ({ ...acc, [key]: true }), {}
  );
  setTouched(allTouched);

  // Valider tout
  const formErrors = validateForm();
  setErrors(formErrors);

  // Si erreurs, ne pas soumettre
  if (Object.keys(formErrors).length > 0) {
    return;
  }
```

```
    // Simuler l'envoi
    setIsSubmitting(true);
    await new Promise(resolve => setTimeout(resolve, 1500));
    setIsSubmitting(false);
    setSubmitSuccess(true);
  }

  // Vérifier si le formulaire est valide
  const isFormValid = Object.keys(validateForm()).length === 0;

  if (submitSuccess) {
    return (
      <div className="signup-success">
        <h2>☑ Inscription réussie !</h2>
        <p>Bienvenue, {formData.name} !</p>
      </div>
    );
  }

  return (
    <form onSubmit={handleSubmit} className="signup-form">
      <h2>Créer un compte</h2>

      {/* Champ Nom */}
      <div className={`form-group ${errors.name && touched.name ? 'has-error' :
''}`}>
        <label htmlFor="name">Nom</label>
        <input
          type="text"
          id="name"
          name="name"
          value={formData.name}
          onChange={handleChange}
          onBlur={handleBlur}
          placeholder="Votre nom"
        />
        {errors.name && touched.name && (
          <span className="error-message">{errors.name}</span>
        )}
      </div>

      {/* Champ Email */}
      <div className={`form-group ${errors.email && touched.email ? 'has-error' :
''}`}>
        <label htmlFor="email">Email</label>
        <input
          type="email"
          id="email"
          name="email"
          value={formData.email}
          onChange={handleChange}
          onBlur={handleBlur}
          placeholder="votre@email.com"
        />
```

```
        {errors.email && touched.email && (
          <span className="error-message">{errors.email}</span>
        )}
      </div>

      {/* Champ Mot de passe */}
      <div className={`form-group ${errors.password && touched.password ? 'has-
error' : ''}`}>
        <label htmlFor="password">Mot de passe</label>
        <input
          type="password"
          id="password"
          name="password"
          value={formData.password}
          onChange={handleChange}
          onBlur={handleBlur}
          placeholder="Minimum 8 caractères"
        />
        {errors.password && touched.password && (
          <span className="error-message">{errors.password}</span>
        )}
        <PasswordStrength password={formData.password} />
      </div>

      {/* Champ Confirmation */}
      <div className={`form-group ${errors.confirmPassword &&
touched.confirmPassword ? 'has-error' : ''}`}>
        <label htmlFor="confirmPassword">Confirmer le mot de passe</label>
        <input
          type="password"
          id="confirmPassword"
          name="confirmPassword"
          value={formData.confirmPassword}
          onChange={handleChange}
          onBlur={handleBlur}
          placeholder="Répétez le mot de passe"
        />
        {errors.confirmPassword && touched.confirmPassword && (
          <span className="error-message">{errors.confirmPassword}</span>
        )}
      </div>

      <button
        type="submit"
        className="submit-btn"
        disabled={isSubmitting}
      >
        {isSubmitting ? 'Inscription...' : 'S\'inscrire'}
      </button>
    </form>
  );
}

// Composant indicateur de force du mot de passe
```

```
function PasswordStrength({ password }) {
  const getStrength = () => {
    let score = 0;
    if (password.length >= 8) score++;
    if (password.length >= 12) score++;
    if (/[A-Z]/.test(password)) score++;
    if (/[0-9]/.test(password)) score++;
    if (/[^A-Za-z0-9]/.test(password)) score++;
    return score;
  };

  const strength = getStrength();
  const labels = ['Très faible', 'Faible', 'Moyen', 'Fort', 'Très fort'];
  const colors = ['#ef4444', '#f97316', '#eab308', '#22c55e', '#16a34a'];

  if (!password) return null;

  return (
    <div className="password-strength">
      <div className="strength-bars">
        {[1, 2, 3, 4, 5].map(level => (
          <div
            key={level}
            className="strength-bar"
            style={{
              backgroundColor: level <= strength ? colors[strength - 1] :
'#e5e7eb'
            }}
          />
        ))}
      </div>
      <span className="strength-label" style={{ color: colors[strength - 1] }}>
        {labels[strength - 1] || ''}
      </span>
    </div>
  );
}

export default SignupForm;
```

```
/* SignupForm.css */
.signup-form {
  max-width: 400px;
  margin: 0 auto;
  padding: 2rem;
  background: white;
  border-radius: 12px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
}

.signup-form h2 {
```

```
  margin: 0 0 1.5rem;
  text-align: center;
  color: #1f2937;
}

.form-group {
  margin-bottom: 1rem;
}

.form-group label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: 500;
  color: #374151;
}

.form-group input {
  width: 100%;
  padding: 0.75rem;
  border: 2px solid #e5e7eb;
  border-radius: 8px;
  font-size: 1rem;
  transition: border-color 0.2s;
  box-sizing: border-box;
}

.form-group input:focus {
  outline: none;
  border-color: #3b82f6;
}

.form-group.has-error input {
  border-color: #ef4444;
}

.error-message {
  display: block;
  margin-top: 0.25rem;
  color: #ef4444;
  font-size: 0.875rem;
}

.password-strength {
  margin-top: 0.5rem;
  display: flex;
  align-items: center;
  gap: 0.5rem;
}

.strength-bars {
  display: flex;
  gap: 4px;
  flex: 1;
}
```

```css
.strength-bar {
  height: 4px;
  flex: 1;
  border-radius: 2px;
  transition: background-color 0.2s;
}

.strength-label {
  font-size: 0.75rem;
  font-weight: 500;
}

.submit-btn {
  width: 100%;
  padding: 0.75rem;
  margin-top: 1rem;
  background: #3b82f6;
  color: white;
  border: none;
  border-radius: 8px;
  font-size: 1rem;
  font-weight: 500;
  cursor: pointer;
  transition: background 0.2s;
}

.submit-btn:hover:not(:disabled) {
  background: #2563eb;
}

.submit-btn:disabled {
  background: #9ca3af;
  cursor: not-allowed;
}

.signup-success {
  text-align: center;
  padding: 3rem;
  background: #d1fae5;
  border-radius: 12px;
}

.signup-success h2 {
  color: #065f46;
  margin: 0 0 0.5rem;
}
```