# Exercice 9 - Formulaire de contact

## Objectif

Créer un formulaire complet avec inputs contrôlés et validation.

## Concepts abordés

- Inputs contrôlés (`value` + `onChange`)
- Gestion d'un objet dans le state
- Validation de formulaire
- Soumission et `preventDefault()`
- useRef pour le focus

---

## Énoncé

Créer un formulaire de contact avec :

1. Champ Nom (requis, min 2 caractères)
2. Champ Email (requis, format email valide)
3. Champ Message (requis, min 10 caractères)
4. Checkbox "Accepter les conditions" (requis)
5. Bouton Envoyer (désactivé si formulaire invalide)

### Comportement attendu

- Validation en temps réel
- Messages d'erreur sous chaque champ
- Focus sur le premier champ en erreur à la soumission
- Réinitialisation après envoi réussi

---

## Correction

```jsx
// src/exercices/ex09-formulaire/FormulaireContact.jsx

import { useState, useRef } from 'react';
import './FormulaireContact.css';

function FormulaireContact() {
  // ═══════════════════════════════════════════════════════════
  // REFS
  // Pour donner le focus aux champs en erreur
  // ═══════════════════════════════════════════════════════════
  const nomRef = useRef(null);
  const emailRef = useRef(null);
  const messageRef = useRef(null);
```

```
// ═══════════════════════════════════════════════════════════════
// STATE DU FORMULAIRE
// On regroupe tous les champs dans un seul objet
// ═══════════════════════════════════════════════════════════════
const [form, setForm] = useState({
  nom: '',
  email: '',
  message: '',
  accepteConditions: false
});

// State pour les erreurs de validation
const [errors, setErrors] = useState({});

// State pour le feedback utilisateur
const [isSubmitting, setIsSubmitting] = useState(false);
const [submitSuccess, setSubmitSuccess] = useState(false);

// ═══════════════════════════════════════════════════════════════
// HANDLER GÉNÉRIQUE POUR TOUS LES CHAMPS
//
// Grâce à l'attribut "name" sur chaque input, on peut utiliser
// un seul handler pour tous les champs.
// ═══════════════════════════════════════════════════════════════
function handleChange(e) {
  const { name, value, type, checked } = e.target;

  // Pour les checkbox, on utilise "checked" au lieu de "value"
  const newValue = type === 'checkbox' ? checked : value;

  setForm(prev => ({
    ...prev,
    [name]: newValue  // [name] = computed property name
  }));

  // Effacer l'erreur quand l'utilisateur corrige
  if (errors[name]) {
    setErrors(prev => ({
      ...prev,
      [name]: null
    }));
  }

  // Réinitialiser le message de succès si l'utilisateur modifie
  if (submitSuccess) {
    setSubmitSuccess(false);
  }
}

// ═══════════════════════════════════════════════════════════════
// VALIDATION
// Retourne un objet avec les erreurs trouvées
// ═══════════════════════════════════════════════════════════════
```

```
  function validate() {
    const newErrors = {};

    // Validation du nom
    if (!form.nom.trim()) {
      newErrors.nom = 'Le nom est requis';
    } else if (form.nom.trim().length < 2) {
      newErrors.nom = 'Le nom doit contenir au moins 2 caractères';
    }

    // Validation de l'email
    if (!form.email.trim()) {
      newErrors.email = "L'email est requis";
    } else if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(form.email)) {
      newErrors.email = "L'email n'est pas valide";
    }

    // Validation du message
    if (!form.message.trim()) {
      newErrors.message = 'Le message est requis';
    } else if (form.message.trim().length < 10) {
      newErrors.message = 'Le message doit contenir au moins 10 caractères';
    }

    // Validation des conditions
    if (!form.accepteConditions) {
      newErrors.accepteConditions = 'Vous devez accepter les conditions';
    }

    return newErrors;
  }

  // ═══════════════════════════════════════════════════════════
  // SOUMISSION
  // ═══════════════════════════════════════════════════════════
  function handleSubmit(e) {
    // Empêcher le rechargement de la page
    e.preventDefault();

    // Valider
    const validationErrors = validate();

    // S'il y a des erreurs, les afficher et focus le premier champ
    if (Object.keys(validationErrors).length > 0) {
      setErrors(validationErrors);

      // Focus sur le premier champ en erreur
      if (validationErrors.nom) {
        nomRef.current.focus();
      } else if (validationErrors.email) {
        emailRef.current.focus();
      } else if (validationErrors.message) {
        messageRef.current.focus();
      }
```

```
      return;
    }

    // Simulation d'envoi
    setIsSubmitting(true);

    // Simuler un appel API
    setTimeout(() => {
      console.log('Formulaire envoyé :', form);

      // Succès : réinitialiser le formulaire
      setForm({
        nom: '',
        email: '',
        message: '',
        accepteConditions: false
      });
      setErrors({});
      setIsSubmitting(false);
      setSubmitSuccess(true);
    }, 1000);
  }

  // ═══════════════════════════════════════════════════════════
  // VÉRIFICATION SI LE FORMULAIRE EST VALIDE (pour le bouton)
  // ═══════════════════════════════════════════════════════════
  const isFormValid =
    form.nom.trim().length >= 2 &&
    /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(form.email) &&
    form.message.trim().length >= 10 &&
    form.accepteConditions;

  // ═══════════════════════════════════════════════════════════
  // RENDU
  // ═══════════════════════════════════════════════════════════
  return (
    <div className="form-container">
      <h2>Contactez-nous</h2>

      {submitSuccess && (
        <div className="success-message">
          Votre message a été envoyé avec succès !
        </div>
      )}

      <form onSubmit={handleSubmit} noValidate>
        {/* ═══════════════════════════════════════════════
            CHAMP NOM
        ═══════════════════════════════════════════════ */}
        <div className="form-group">
          <label htmlFor="nom">Nom *</label>
          <input
            ref={nomRef}
```

```
          type="text"
          id="nom"
          name="nom"
          value={form.nom}
          onChange={handleChange}
          className={errors.nom ? 'input-error' : ''}
          placeholder="Votre nom"
        />
        {errors.nom && (
          <span className="error-message">{errors.nom}</span>
        )}
      </div>

      {/* ═══════════════════════════════════════════════════
          CHAMP EMAIL
      ═══════════════════════════════════════════════════ */}
      <div className="form-group">
        <label htmlFor="email">Email *</label>
        <input
          ref={emailRef}
          type="email"
          id="email"
          name="email"
          value={form.email}
          onChange={handleChange}
          className={errors.email ? 'input-error' : ''}
          placeholder="votre@email.com"
        />
        {errors.email && (
          <span className="error-message">{errors.email}</span>
        )}
      </div>

      {/* ═══════════════════════════════════════════════════
          CHAMP MESSAGE
      ═══════════════════════════════════════════════════ */}
      <div className="form-group">
        <label htmlFor="message">Message *</label>
        <textarea
          ref={messageRef}
          id="message"
          name="message"
          value={form.message}
          onChange={handleChange}
          className={errors.message ? 'input-error' : ''}
          placeholder="Votre message (min. 10 caractères)"
          rows={5}
        />
        {errors.message && (
          <span className="error-message">{errors.message}</span>
        )}
        <span className="char-count">
          {form.message.length} / 10 caractères minimum
        </span>
```

```
        </div>

        {/* ═══════════════════════════════════════════════════
            CHECKBOX CONDITIONS
        ═══════════════════════════════════════════════════ */}
        <div className="form-group checkbox-group">
          <label>
            <input
              type="checkbox"
              name="accepteConditions"
              checked={form.accepteConditions}
              onChange={handleChange}
            />
            <span>J'accepte les conditions d'utilisation *</span>
          </label>
          {errors.accepteConditions && (
            <span className="error-message">{errors.accepteConditions}</span>
          )}
        </div>

        {/* ═══════════════════════════════════════════════════
            BOUTON SUBMIT
        ═══════════════════════════════════════════════════ */}
        <button
          type="submit"
          disabled={!isFormValid || isSubmitting}
          className="submit-btn"
        >
          {isSubmitting ? 'Envoi en cours...' : 'Envoyer'}
        </button>
      </form>
    </div>
  );
}

export default FormulaireContact;
```

```
/* src/exercices/ex09-formulaire/FormulaireContact.css */

.form-container {
  max-width: 500px;
  margin: 0 auto;
  padding: 2rem;
}

.form-group {
  margin-bottom: 1.5rem;
}

.form-group label {
  display: block;
```

```
  margin-bottom: 0.5rem;
  font-weight: 500;
}

.form-group input[type="text"],
.form-group input[type="email"],
.form-group textarea {
  width: 100%;
  padding: 0.75rem;
  font-size: 1rem;
  border: 1px solid #d1d5db;
  border-radius: 4px;
  transition: border-color 0.2s;
}

.form-group input:focus,
.form-group textarea:focus {
  outline: none;
  border-color: #2563eb;
  box-shadow: 0 0 0 3px rgba(37, 99, 235, 0.1);
}

.input-error {
  border-color: #dc2626 !important;
}

.error-message {
  display: block;
  margin-top: 0.25rem;
  color: #dc2626;
  font-size: 0.875rem;
}

.char-count {
  display: block;
  margin-top: 0.25rem;
  color: #6b7280;
  font-size: 0.75rem;
}

.checkbox-group label {
  display: flex;
  align-items: center;
  gap: 0.5rem;
  cursor: pointer;
}

.submit-btn {
  width: 100%;
  padding: 0.75rem;
  font-size: 1rem;
  font-weight: 500;
  color: white;
  background: #2563eb;
```

```
  border: none;
  border-radius: 4px;
  cursor: pointer;
  transition: background 0.2s;
}

.submit-btn:hover:not(:disabled) {
  background: #1d4ed8;
}

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

.success-message {
  padding: 1rem;
  margin-bottom: 1rem;
  background: #d1fae5;
  color: #065f46;
  border-radius: 4px;
}
```

---

## Points clés

### Handler générique avec computed property name

```
// L'attribut name de l'input devient la clé dans le state
function handleChange(e) {
  const { name, value } = e.target;
  setForm(prev => ({
    ...prev,
    [name]: value  // [name] = computed property name
  }));
}

// Fonctionne car chaque input a un name correspondant au state
<input name="email" value={form.email} onChange={handleChange} />
<input name="nom" value={form.nom} onChange={handleChange} />
```

### Checkbox : checked vs value

```
// Pour les checkbox, utiliser "checked" pas "value"
<input
  type="checkbox"
  name="accepte"
  checked={form.accepte}  // PAS value={form.accepte}
  onChange={handleChange}
```

```
/>

// Et dans le handler
const newValue = type === 'checkbox' ? checked : value;
```

### Toujours e.preventDefault() sur onSubmit

```
function handleSubmit(e) {
  e.preventDefault();  // OBLIGATOIRE sinon la page recharge
  // ...
}
```