Événements et formulaires

Gestion des interactions utilisateur

- Événements
- Formulaires contrôlés

Événements

Gestion des événements

React utilise des événements synthétiques (`SyntheticEvent`) qui wrappent les événements natifs du navigateur.

Syntaxe

```
// Passer une référence de fonction (pas d'appel !)
<button onClick={handleClick}>Clic</button>

// Arrow function inline (pour passer des arguments)
<button onClick={() => handleClick(id)}>Supprimer</button>

// Avec l'objet event
function handleClick(e) {
  e.preventDefault();  // Empêcher le comportement par défaut
  e.stopPropagation(); // Stopper la propagation
  console.log(e.target.value);
}
```

Événements courants

Événement	Description
`onClick`	Clic souris
`onChange`	Valeur d'input modifiée
`onSubmit`	Soumission de formulaire
`onKeyDown` / `onKeyUp`	Touches clavier
`onFocus` / `onBlur`	Focus / perte de focus
`onMouseEnter` / `onMouseLeave`	Survol souris
`onScroll`	Défilement

Délégation d'événement

```
// Gérer les clics de plusieurs enfants dans le parent
function Liste({ items, onDelete }) {
  return (
    <ul onClick={(e) => {
      const id = e.target.closest('li')?.dataset.id;
      if (id) onDelete(id);
    }}>
      {items.map(item => (
        <li key={item.id} data-id={item.id}>{item.nom}</li>
      ))}
    </ul>
  );
}
```

Formulaires contrôlés

Formulaires contrôlés

Dans un formulaire contrôlé, React gère l'état de chaque champ via le state. C'est le pattern recommandé.

Input contrôlé simple

```
function Formulaire() {
  const [nom, setNom] = useState("");

  return (
    <input
      value={nom}
      onChange={(e) => setNom(e.target.value)}
      placeholder="Votre nom"
    />
  );
}
```

Formulaire complet

```
function Inscription() {
  const [form, setForm] = useState({
    nom: "", email: "", password: "", role: "user"
  });
  const [errors, setErrors] = useState({});

  const handleChange = (e) => {
    const { name, value, type, checked } = e.target;
    setForm(prev => ({
      ...prev,
      [name]: type === 'checkbox' ? checked : value
    }));
```

```
  };

  const validate = () => {
    const newErrors = {};
    if (!form.nom) newErrors.nom = "Nom requis";
    if (!form.email.includes('@')) newErrors.email = "Email invalide";
    if (form.password.length < 6) newErrors.password = "6 caractères min";
    return newErrors;
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const newErrors = validate();
    if (Object.keys(newErrors).length > 0) {
      setErrors(newErrors);
      return;
    }
    console.log("Soumission :", form);
  };

  return (
    <form onSubmit={handleSubmit}>
      <div>
        <input name="nom" value={form.nom} onChange={handleChange} />
        {errors.nom && <span>{errors.nom}</span>}
      </div>
      <div>
        <select name="role" value={form.role} onChange={handleChange}>
          <option value="user">Utilisateur</option>
          <option value="admin">Admin</option>
        </select>
      </div>
      <button type="submit">S'inscrire</button>
    </form>
  );
}
```

Bibliothèques de formulaires

Pour des formulaires complexes, préférer :

- **React Hook Form** — performant, peu de re-renders, validation facile
- **Formik** — complet, plus verbeux
- **Zod** — validation de schéma TypeScript-first, souvent couplé avec RHF