Cheat Sheet - Formulaires React

Deux approches : Contrôlé vs Non contrôlé

	Contrôlé	Non contrôlé
Source de vérité	React (useState)	Le DOM
Accès à la valeur	Via le state	Via useRef
Mise à jour	À chaque frappe	À la soumission
Validation temps réel	Facile	Difficile
Complexité	Plus de code	Moins de code
Usage recommandé	90% des cas	Cas simples, intégration libs

Input Contrôlé (Recommandé)

Principe

React contrôle la valeur : `value` + `onChange`

```
function ControlledInput() {
  const [email, setEmail] = useState("");

  return (
    <input
      type="email"
      value={email}                                    // Valeur = state
      onChange={(e) => setEmail(e.target.value)}      // Mise à jour du state
    />
  );
}
```

Tous les types d'inputs

```
function CompleteForm() {
  const [form, setForm] = useState({
    name: "",
    email: "",
    age: "",
    bio: "",
    country: "france",
    newsletter: false,
    gender: "",
  });

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

```
return (
  <form>
    {/* Text */}
    <input
      type="text"
      name="name"
      value={form.name}
      onChange={handleChange}
    />

    {/* Email */}
    <input
      type="email"
      name="email"
      value={form.email}
      onChange={handleChange}
    />

    {/* Number */}
    <input
      type="number"
      name="age"
      value={form.age}
      onChange={handleChange}
    />

    {/* Textarea */}
    <textarea
      name="bio"
      value={form.bio}
      onChange={handleChange}
    />

    {/* Select */}
    <select
      name="country"
      value={form.country}
      onChange={handleChange}
    >
      <option value="france">France</option>
      <option value="belgique">Belgique</option>
      <option value="suisse">Suisse</option>
    </select>

    {/* Checkbox */}
    <input
      type="checkbox"
      name="newsletter"
      checked={form.newsletter}           // checked, pas value !
      onChange={handleChange}
    />

    {/* Radio */}
    <input
      type="radio"
      name="gender"
      value="homme"
      checked={form.gender === "homme"}
      onChange={handleChange}
    />
    <input
      type="radio"
      name="gender"
      value="femme"
      checked={form.gender === "femme"}
```

```
        onChange={handleChange}
      />
    </form>
  );
}
```

Input Non Contrôlé

Principe

Le DOM garde la valeur, on lit via `ref`

```
function UncontrolledInput() {
  const inputRef = useRef(null);

  function handleSubmit(e) {
    e.preventDefault();
    const value = inputRef.current.value;  // Lecture au moment voulu
    console.log(value);
  }

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        ref={inputRef}
        defaultValue="initial@email.com"   // defaultValue, pas value !
      />
      <button type="submit">Envoyer</button>
    </form>
  );
}
```

Attributs `value` vs `defaultValue`

Contrôlé	Non contrôlé
`value={state}`	`defaultValue="valeur"`
`checked={state}`	`defaultChecked={true}`

Soumission de formulaire

Pattern complet

```
function LoginForm() {
  const [form, setForm] = useState({ email: "", password: "" });
  const [errors, setErrors] = useState({});
  const [isSubmitting, setIsSubmitting] = useState(false);

  function handleChange(e) {
    const { name, value } = e.target;
    setForm(prev => ({ ...prev, [name]: value }));

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

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

  if (!form.email) {
    newErrors.email = "L'email est requis";
  } else if (!/\S+@\S+\.\S+/.test(form.email)) {
    newErrors.email = "Email invalide";
  }

  if (!form.password) {
    newErrors.password = "Le mot de passe est requis";
  } else if (form.password.length < 6) {
    newErrors.password = "Minimum 6 caractères";
  }

  return newErrors;
}

async function handleSubmit(e) {
  e.preventDefault();

  const validationErrors = validate();
  if (Object.keys(validationErrors).length > 0) {
    setErrors(validationErrors);
    return;
  }

  setIsSubmitting(true);
  try {
    await api.login(form);
    // Redirection ou succès
  } catch (err) {
    setErrors({ submit: err.message });
  } finally {
    setIsSubmitting(false);
  }
}

return (
  <form onSubmit={handleSubmit}>
    <div>
      <input
        type="email"
        name="email"
        value={form.email}
        onChange={handleChange}
        className={errors.email ? "error" : ""}
      />
      {errors.email && <span className="error-msg">{errors.email}</span>}
    </div>

    <div>
      <input
        type="password"
        name="password"
        value={form.password}
        onChange={handleChange}
        className={errors.password ? "error" : ""}
      />
      {errors.password && <span className="error-msg">{errors.password}</span>}
    </div>

    {errors.submit && <p className="error-msg">{errors.submit}</p>}
```

```
      <button type="submit" disabled={isSubmitting}>
        {isSubmitting ? "Envoi..." : "Se connecter"}
      </button>
    </form>
  );
}
```

Patterns utiles

Bouton désactivé si formulaire incomplet

```
const isDisabled = !form.email || !form.password || form.password.length < 6;

<button type="submit" disabled={isDisabled}>
  Envoyer
</button>
```

Reset du formulaire

```
const initialState = { email: "", password: "" };
const [form, setForm] = useState(initialState);

function handleReset() {
  setForm(initialState);
  setErrors({});
}

<button type="button" onClick={handleReset}>Réinitialiser</button>
```

Focus sur le premier champ en erreur

```
const emailRef = useRef(null);
const passwordRef = useRef(null);

function handleSubmit(e) {
  e.preventDefault();
  const errors = validate();

  if (errors.email) {
    emailRef.current.focus();
  } else if (errors.password) {
    passwordRef.current.focus();
  }
}

<input ref={emailRef} ... />
<input ref={passwordRef} ... />
```

Tableau récapitulatif des événements

Élément	Événement	Accès à la valeur
`<input type="text">`	`onChange`	`e.target.value`
`<input type="number">`	`onChange`	`e.target.value` (string!)
`<input type="checkbox">`	`onChange`	`e.target.checked`

Élément	Événement	Accès à la valeur
`<input type="radio">`	`onChange`	`e.target.value`
`<select>`	`onChange`	`e.target.value`
`<textarea>`	`onChange`	`e.target.value`
`<form>`	`onSubmit`	-

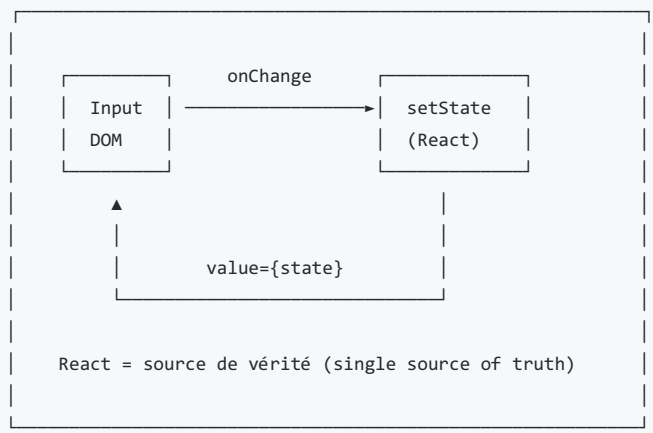
Piège : input number retourne un string

```
const [age, setAge] = useState(0);

// e.target.value est TOUJOURS un string
<input
  type="number"
  value={age}
  onChange={(e) => setAge(Number(e.target.value))}  // Conversion !
/>

// Alternative : parseInt ou +
onChange={(e) => setAge(parseInt(e.target.value, 10))}
onChange={(e) => setAge(+e.target.value)}
```

Schéma mental : Formulaire contrôlé



Checklist Formulaires

- [ ] `onSubmit` sur le `<form>`, pas `onClick` sur le bouton
- [ ] `e.preventDefault()` dans le handler de soumission
- [ ] Validation avant soumission
- [ ] Gestion des erreurs (affichage + focus)
- [ ] État de chargement (`isSubmitting`)
- [ ] Bouton désactivé pendant le chargement
- [ ] Checkbox : `checked` au lieu de `value`
- [ ] Input number : convertir en Number