

Atelier 02.05 Formulaires

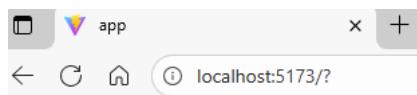
Exercice 1 — Champ Texte : Nom

Objectif

Créer un champ **contrôlé** et stocker la valeur dans l'état.

Instruction

Créer un input texte **name** et afficher sa valeur.



Ajouter un Contact

Nom : ABBASSI

Valeur : ABBASSI

Correction

```
// src/components/FormulaireContact.jsx
import { useState } from "react";

function FormulaireContact() {
  const [form, setForm] = useState({});

  const handleChange = (e) => {
    /* Les crochets [] servent à utiliser une variable comme nom de propriété. */
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  return (
    <div>
      <h2>Ajouter un Contact</h2>

      <label>Nom :</label>
      <input type="text" name="name" onChange={handleChange} />

      <p>Valeur : {form.name}</p>
    </div>
  );
}

export default FormulaireContact;
```

Exercice 2 — Ajouter Champ Email

Objectif

Gérer plusieurs champs dans un seul objet.

Instruction

Ajouter un input email.

Correction

```
// src/components/FormulaireContact.jsx
import { useState } from "react";

function FormulaireContact() {
  const [form, setForm] = useState({});

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

  return (
    <div>
      <h2>Ajouter un Contact</h2>

      <label>Nom :</label>
      <input type="text" name="name" onChange={handleChange} />

      <label>Email :</label>
      <input type="email" name="email" onChange={handleChange} />

      <pre>{JSON.stringify(form, null, 2)}</pre>
    </div>
  );
}

export default FormulaireContact;
```

Exercice 3 — Champ Téléphone**Objectif**

Ajouter un champ phone.

Correction

```
<label>Téléphone :</label>
<input type="text" name="phone" onChange={handleChange} />
```

Exercice 4 — Champ Date (Birthday)**Objectif**

Ajouter un champ de type date.

Correction

```
<label>Date de naissance :</label>
<input type="date" name="birthdate" onChange={handleChange} />
```

Exercice 5 — Champ Select (country)**Objectif**

Ajouter un choix parmi des pays.

Correction

```
<label>Pays :</label>
<select name="country" onChange={handleChange}>
  <option value="">-- Choisir --</option>
  <option value="Tunisie">Tunisie</option>
  <option value="France">France</option>
  <option value="Algérie">Algérie</option>
</select>
```

Exercice 6 — Textarea (notes)**Objectif**

Saisir du texte sur plusieurs lignes.

Correction

```
<label>Notes :</label>
<textarea name="notes" onChange={handleChange}></textarea>
```

Exercice 7 — Fichier Image (Avatar)**Objectif**

Importer et afficher une image.

Correction

```
const handleAvatar = (e) => {
  const file = e.target.files[0];
  setForm({ ...form, avatar: file, avatarPreview: URL.createObjectURL(file) });
};

<label>Photo de profil :</label>
<input type="file" accept="image/*" onChange={handleAvatar} />

{form.avatarPreview && <img src={form.avatarPreview} width="120" />}
```

Exercice 8 — Fichier PDF (Cv)**Objectif**

Uploader un document PDF et l'enregistrer dans le state.

Correction

```
const handlePDF = (e) => {
  setForm({ ...form, cv: e.target.files[0].file });
};

<label>CV (PDF) :</label>
<input type="file" accept="application/pdf" onChange={handlePDF} />
```

Exercice 9 : ajouter les balise from et submit

```
.....
<h2>Ajouter un Contact</h2>
<form onSubmit={handleSubmit}>
.....
<button type="submit">Envoyer</button>
  </form>
/* Visualization */
<pre>{JSON.stringify(form, null, 2)}</pre>
.....
```

```
// Empêcher le rechargement et afficher les données envoyées
const handleSubmit = (e) => {
  e.preventDefault();
  console.log("Données envoyées :", form);
  alert("Contact ajouté avec succès !");
};
```

Formulaire Final : Ajouter un Contact

```
// src/components/FormulaireContact.jsx
import { useState } from "react";
import './FormulaireContact.css';

function FormulaireContact() {
    const [form, setForm] = useState({});

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

    const handleAvatar = (e) => {
        const file = e.target.files[0];
        setForm({ ...form, avatar: file, avatarPreview: URL.createObjectURL(file) });
    };
    const handlePDF = (e) => {
        setForm({ ...form, cv: e.target.files[0].name });
        console.log("Fichier sélectionné →", e.target.files[0].name);
    };
// Empêcher le rechargement et afficher les données envoyées
    const handleSubmit = (e) => {
        e.preventDefault();
        console.log("Données envoyées :", form);
        alert("Contact ajouté avec succès !");
    };

    return (
        <div>
            <h2>Ajouter un Contact</h2>
            <form onSubmit={handleSubmit}>

                <label>Nom :</label>
                <input type="text" name="name" onChange={handleChange} />
                <label>Email :</label>
                <input type="email" name="email" onChange={handleChange} />
                <label>Téléphone :</label>
                <input type="text" name="phone" onChange={handleChange} />
                <label>Date de naissance :</label>
                <input type="date" name="birthdate" onChange={handleChange} />
                <label>Pays :</label>
                <select name="country" onChange={handleChange}>
                    <option value="">-- Choisir --</option>
                    <option value="Tunisie">Tunisie</option>
                    <option value="France">France</option>
                    <option value="Algérie">Algérie</option>
                </select>
                <label>Notes :</label>
                <textarea name="notes" onChange={handleChange}></textarea>

                <label>Photo de profil :</label>
                <input type="file" accept="image/*" onChange={handleAvatar} />

                {form.avatarPreview && <img src={form.avatarPreview} width="120" />}

                <label>CV (PDF) :</label>
                <input type="file" accept="application/pdf" onChange={handlePDF} />

            <button type="submit">Ajouter</button>
        </form>
        <pre>{JSON.stringify(form, null, 2)}</pre>
    );
}

export default FormulaireContact;
```

Feuille de Style

```
/* src/components/FormulaireContact.css */
```

```
/* Conteneur général */
div {
    max-width: 450px;
    margin: 40px auto;
    padding: 25px 30px;
    border-radius: 12px;
    background: #ffffff;
    box-shadow: 0px 4px 12px rgba(0,0,0,0.12);
    font-family: "Segoe UI", Arial, sans-serif;
}

/* Titre */
h2 {
    text-align: center;
    margin-bottom: 20px;
    color: #333;
}

/* Formulaire */
form {
    display: flex;
    flex-direction: column;
    gap: 14px;
}

/* Label */
label {
    font-weight: 600;
    color: #444;
}

/* Inputs, Select, Textarea */
input[type="text"],
input[type="email"],
input[type="date"],
input[type="file"],
input[type="number"],
textarea,
select {
    padding: 8px 10px;
    border: 1px solid #bbb;
    border-radius: 6px;
    transition: 0.3s;
    font-size: 14px;
}

/* Effet focus */
input:focus,
textarea:focus,
select:focus {
    border-color: #007bff;
    outline: none;
    box-shadow: 0px 0px 4px rgba(0,123,255,0.4);
}

/* Textarea */
textarea {
    min-height: 70px;
    resize: vertical;
}

/* Bouton */
button {
    padding: 10px;
    border: none;
    border-radius: 6px;
    background: #007bff;
    color: white;
    font-size: 15px;
    cursor: pointer;
    margin-top: 10px;
    transition: 0.3s;
}
```

```
button:hover {  
    background: #0056c5;  
}  
  
/* Image Preview */  
img {  
    display: block;  
    margin-top: 8px;  
    border-radius: 8px;  
    border: 1px solid #ddd;  
}  
  
/* Zone JSON Debug */  
pre {  
    background: #f8f8f8;  
    padding: 12px;  
    margin-top: 20px;  
    border-radius: 6px;  
    font-size: 13px;  
    overflow-x: auto;  
    border: 1px solid #e0e0e0;  
}
```

Ajouter un Contact

Nom :

Email :

Téléphone :

Date de naissance :

CALENDAR

Pays :

Notes :

Test de notes

Photo de profil :

Choisir un fichier1580159721520.jpg



CV (PDF) :

Choisir un fichierArticle 04.pdf

Ajouter

```
{  
    "name": "Kamel ABBASSI",  
    "email": "abbassi.kamel@gmail.com",  
    "phone": "26388202",  
    "birthdate": "2025-10-29",  
    "country": "Tunisie",  
    "notes": "Test de notes",  
    "avatar": {},  
    "avatarPreview": "blob:http://localhost:5173/5ec47a",  
    "cv": "Article 04.pdf"  
}
```

Exercice 5 — Comprendre les fonctions de traitement de données

Objectif

Découvrir les fonctions JavaScript utiles pour manipuler les données des formulaires avant de les envoyer ou de les valider.

Instructions à suivre

1. Crée un fichier src/components/JsBasics.jsx.
2. Dans ce composant, crée plusieurs exemples simples affichés à l'écran :
 - trim() : supprimer les espaces inutiles.
 - includes() : vérifier qu'une chaîne contient un caractère.
 - length : compter le nombre de caractères.
 - JSON.stringify() : convertir un objet en texte JSON.
 - JSON.parse() : reconvertisir le texte JSON en objet.
 - localStorage.setItem() / localStorage.getItem() : stocker et relire une donnée localement.
3. Chaque exemple doit être affiché sous forme de paragraphe clair.

Correction

```
// src/components/JsBasics.jsx
export default function JsBasics() {
  const texte = " ReactJS ";
  const trimmed = texte.trim(); // supprime espaces
  const contientR = trimmed.includes("R"); // vérifie si "R" est présent
  const longueur = trimmed.length; // compte les caractères

  const utilisateur = { nom: "Ali", age: 25 };
  const jsonTexte = JSON.stringify(utilisateur); // objet -> texte
  const objet = JSON.parse(jsonTexte); // texte -> objet

  localStorage.setItem("nom", "Kamel");
  const nomStocke = localStorage.getItem("nom");

  return (
    <div className="jsbasics">
      <h3>Exemples de fonctions utiles</h3>
      <p>Texte original : "{texte}"</p>
      <p>Après trim() : "{trimmed}"</p>
      <p>Contient "R" ? {contientR ? "Oui" : "Non"}</p>
      <p>Longueur du texte : {longueur}</p>
      <p>Objet JSON.stringify : {jsonTexte}</p>
      <p>Objet JSON.parse : {objet.nom} - {objet.age} ans</p>
      <p>Valeur dans localStorage : {nomStocke}</p>
    </div>
  );
}
```

Feuille de style src/JsBasics.css

```
.jsbasics {
  font-family: Arial, sans-serif;
```

```
background-color: #f7f7ff;
border: 1px solid #ddd;
padding: 15px;
border-radius: 8px;
max-width: 500px;
margin: 20px auto;
}
.jsbasics h3 {
color: #333;
text-align: center;
}
.jsbasics p {
margin: 6px 0;
font-size: 14px;
}
```

Exercice 6 — Créer un formulaire contrôlé

Objectif

Créer un formulaire React contrôlé et récupérer les données saisies.

Instructions

1. Crée src/components/FormulaireSimple.jsx.
2. Le formulaire contient deux champs : nom, email et un bouton Envoyer.
3. Utilise useState pour contrôler les champs.
4. À la soumission, affiche les valeurs dans la console.

Correction

```
// src/components/FormulaireSimple.jsx
import { useState } from "react";
import "./FormulaireSimple.css";

export default function FormulaireSimple() {
  const [nom, setNom] = useState("");
  const [email, setEmail] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Nom :", nom);
    console.log("Email :", email);
  };

  return (
    <form onSubmit={handleSubmit} className="formulaire">
      <label>Nom :</label>
      <input value={nom} onChange={(e) => setNom(e.target.value)} />

      <label>Email :</label>
      <input value={email} onChange={(e) => setEmail(e.target.value)} />

      <button type="submit">Envoyer</button>
    </form>
  );
}
```

Feuille de style FormulaireSimple.css

```
.formulaire {
  display: flex;
  flex-direction: column;
  width: 300px;
  margin: 20px auto;
  gap: 10px;
  background: #eef;
  padding: 20px;
  border-radius: 8px;
}
.formulaire input {
  padding: 6px;
  border: 1px solid #aaa;
  border-radius: 4px;
}
.formulaire button {
  background-color: #3b82f6;
  color: white;
  border: none;
  padding: 8px;
  border-radius: 4px;
  cursor: pointer;
}
```

Exercice 7 — Validation des champs

Objectif

Valider les données saisies avant soumission.

Instructions

1. Reprends le formulaire précédent.
2. Vérifie :
 - Le nom a au moins 3 caractères (length).
 - L'email contient "@" (includes).
3. Affiche un message d'erreur si invalide.

Correction

```
// src/components/FormValidation.jsx
import { useState } from "react";
import "./FormValidation.css";

export default function FormValidation() {
  const [nom, setNom] = useState("");
  const [email, setEmail] = useState("");
  const [error, setError] = useState("");

  const handleSubmit = (e) => {
    e.preventDefault();

    if (nom.trim().length < 3) {
      setError("Le nom doit contenir au moins 3 caractères.");
      return;
    }

    if (!email.includes("@")) {
      setError("Adresse email invalide.");
      return;
    }
  }
}
```

```

        setError("");
        alert("Formulaire valide !");
    };

    return (
        <form onSubmit={handleSubmit} className="formvalidation">
            <label>Nom :</label>
            <input value={nom} onChange={(e) => setNom(e.target.value)} />
            <label>Email :</label>
            <input value={email} onChange={(e) => setEmail(e.target.value)} />
            {error && <p className="error">{error}</p>}
            <button type="submit">Envoyer</button>
        </form>
    );
}

```

Feuille de style FormValidation.css

```

.formvalidation {
    background: #f9f9ff;
    width: 320px;
    margin: 20px auto;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 6px rgba(0,0,0,0.1);
}
.error {
    color: red;
    font-size: 13px;
}

```

Exercice 8 — Enregistrer les données dans un fichier JSON local

Objectif

Sauvegarder les données validées dans un fichier JSON simulé via localStorage.

Méthode choisie

👉 On stocke dans localStorage (plus simple et compatible navigateur).
public/data.json ne peut pas être écrit directement, donc localStorage agit comme un mini “fichier JSON” local.

Correction

```

// src/components/SaveForm.jsx
import { useState } from "react";
import "./SaveForm.css";

export default function SaveForm() {
    const [nom, setNom] = useState("");
    const [email, setEmail] = useState("");

    const handleSubmit = (e) => {
        e.preventDefault();
        const newUser = { nom, email };
        const data = JSON.parse(localStorage.getItem("users") || "[]");
        data.push(newUser);
        localStorage.setItem("users", JSON.stringify(data));
        alert("Enregistré !");
    };
}

```

```

return (
  <form onSubmit={handleSubmit} className="saveform">
    <label>Nom :</label>
    <input value={nom} onChange={(e) => setNom(e.target.value)} />
    <label>Email :</label>
    <input value={email} onChange={(e) => setEmail(e.target.value)} />
    <button type="submit">Enregistrer</button>
  </form>
);
}

```

Feuille de style SaveForm.css

```

.saveform {
  background: #f0ffff4;
  border: 1px solid #b2f5ea;
  padding: 20px;
  border-radius: 8px;
  width: 320px;
  margin: 20px auto;
}
.saveform button {
  background: #16a34a;
  color: white;
  padding: 8px;
  border: none;
  border-radius: 4px;
}

```

Exercice 9 — Lire, modifier et supprimer des données

Objectif

Créer un petit tableau d'utilisateurs avec modification et suppression.

Correction

```

// src/components/UserManager.jsx
import { useState, useEffect } from "react";
import "./userManager.css";

export default function UserManager() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    const stored = JSON.parse(localStorage.getItem("users") || "[]");
    setUsers(stored);
  }, []);

  const handleDelete = (i) => {
    const updated = users.filter((_, index) => index !== i);
    setUsers(updated);
    localStorage.setItem("users", JSON.stringify(updated));
  };

  const handleEdit = (i) => {
    const newName = prompt("Nouveau nom :", users[i].nom);
    if (newName) {
      const updated = [...users];
      updated[i].nom = newName;
      setUsers(updated);
      localStorage.setItem("users", JSON.stringify(updated));
    }
  };
}

```

```

return (
  <div className="usermanager">
    <h3>Liste des utilisateurs</h3>
    <ul>
      {users.map((u, i) => (
        <li key={i}>
          {u.nom} - {u.email}
          <button onClick={() => handleEdit(i)}>Modifier</button>
          <button onClick={() => handleDelete(i)}>Supprimer</button>
        </li>
      ))}
    </ul>
  </div>
);
}

```

Feuille de style UserManager.css

```

.usermanager {
  width: 400px;
  margin: 20px auto;
  background: #ffffaf;
  border: 1px solid #fcd34d;
  padding: 15px;
  border-radius: 8px;
}
.usermanager button {
  margin-left: 10px;
  border: none;
  padding: 5px 8px;
  border-radius: 4px;
  cursor: pointer;
}

```

Exercice 10 : Formulaire complet avec gestion des données JSON locales

Objectif pédagogique

Apprendre à :

- Créer un **formulaire contrôlé** en ReactJS (avec useState)
- **Valider les champs** selon plusieurs critères (ex : email valide, mot de passe long, nom non vide)
- **Afficher les erreurs de validation** sous les champs
- **Sauvegarder les données dans un fichier JSON local**
- **Lire, modifier et supprimer** des données à partir de ce fichier

Instructions à suivre

Étape 1 : Créer le fichier de données JSON

Crée un dossier src/data/ et ajoute le fichier users.json :

```
[
  {
    "id": 1,
    "name": "Kamel",
    "email": "kamel@example.com",
    "password": "123456"
  }
]
```

```

}
]
```

Étape 2 : Créer le composant UserForm.jsx

```

//Chemin : src/components/UserForm.jsx
import { useState, useEffect } from "react";
import usersData from "../data/users.json";

export default function UserForm() {
  const [users, setUsers] = useState(usersData);
  const [formData, setFormData] = useState({ name: "", email: "", password: "" });
  const [errors, setErrors] = useState({});

  // --- Validation du formulaire ---
  const validate = () => {
    let newErrors = {};
    if (formData.name.trim().length < 3) newErrors.name = "Le nom doit comporter au moins 3 caractères.";
    if (!formData.email.includes("@")) newErrors.email = "L'email doit contenir '@.'";
    if (formData.password.length < 6) newErrors.password = "Le mot de passe doit comporter au moins 6 caractères.";
    return newErrors;
  };

  // --- Soumission du formulaire ---
  const handleSubmit = (e) => {
    e.preventDefault();
    const newErrors = validate();
    if (Object.keys(newErrors).length > 0) {
      setErrors(newErrors);
    } else {
      const newUser = { id: Date.now(), ...formData };
      const updatedUsers = [...users, newUser];
      setUsers(updatedUsers);
      localStorage.setItem("users", JSON.stringify(updatedUsers));
      setFormData({ name: "", email: "", password: "" });
      setErrors({});
    }
  };

  // --- Charger les données depuis localStorage s'il existe ---
  useEffect(() => {
    const storedUsers = localStorage.getItem("users");
    if (storedUsers) {
      setUsers(JSON.parse(storedUsers));
    }
  }, []);

  // --- Supprimer un utilisateur ---
  const deleteUser = (id) => {
    const updated = users.filter((u) => u.id !== id);
    setUsers(updated);
    localStorage.setItem("users", JSON.stringify(updated));
  };

  return (
    <div className="form-container">
      <h2>Gestion des utilisateurs</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label>Nom :</label>
          <input
            type="text"
            value={formData.name}
            onChange={(e) => setFormData({ ...formData, name: e.target.value })}
          />
          {errors.name && <p className="error">{errors.name}</p>}
        </div>
        <div>
          <label>Email :</label>

```

```

<input
  type="email"
  value={formData.email}
  onChange={(e) => setFormData({ ...formData, email: e.target.value })}
/>
{errors.email && <p className="error">{errors.email}</p>}
</div>

<div>
  <label>Mot de passe :</label>
  <input
    type="password"
    value={formData.password}
    onChange={(e) => setFormData({ ...formData, password: e.target.value })}
/>
{errors.password && <p className="error">{errors.password}</p>}
</div>

<button type="submit">Enregistrer</button>
</form>

<h3>Liste des utilisateurs</h3>
<ul>
  {users.map((u) => (
    <li key={u.id}>
      {u.name} ({u.email}){" "}
      <button onClick={() => deleteUser(u.id)}>Supprimer</button>
    </li>
  ))}
</ul>
</div>
);
}

```

Étape 3 : Modifier src/main.jsx

```

import React from "react";
import ReactDOM from "react-dom/client";
import UserForm from "./components/UserForm.jsx";
import "./style.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <UserForm />
  </React.StrictMode>
);

```

Étape 4 : Ajouter une feuille de style style.css

Chemin : src/style.css

```

body {
  font-family: Arial, sans-serif;
  background-color: #f2f2f2;
  margin: 0;
  padding: 0;
}

.form-container {
  width: 400px;
  margin: 40px auto;
  background: white;
  padding: 20px;
  border-radius: 12px;
  box-shadow: 0 0 10px rgba(0,0,0,0.2);
}

```

```
h2, h3 {  
    text-align: center;  
    color: #333;  
}  
  
form div {  
    margin-bottom: 10px;  
}  
  
label {  
    display: block;  
    margin-bottom: 5px;  
}  
  
input {  
    width: 100%;  
    padding: 8px;  
    border: 1px solid #aaa;  
    border-radius: 5px;  
}  
  
button {  
    padding: 8px 15px;  
    background-color: #0077cc;  
    color: white;  
    border: none;  
    border-radius: 6px;  
    cursor: pointer;  
    margin-top: 5px;  
}  
  
button:hover {  
    background-color: #005fa3;  
}  
  
.error {  
  
    color: red;  
    font-size: 13px;  
    margin-top: 2px;  
}  
  
ul {  
    list-style-type: none;  
    padding: 0;  
}  
  
li {  
    background: #e6f0ff;  
    margin: 5px 0;  
    padding: 8px;  
    border-radius: 5px;  
}
```

Résultat attendu

Lorsqu'un participant :

- remplit un nom, email et mot de passe valides, puis clique sur **Enregistrer**
👉 Les données s'ajoutent dans la liste et sont sauvegardées dans le **localStorage**.
- actualise la page, la liste reste affichée (chargée depuis le localStorage).
- clique sur **Supprimer**, l'utilisateur est effacé de la liste et du stockage.

Correction et explications

- useState gère les données du formulaire et la liste d'utilisateurs.
- validate() vérifie chaque champ et retourne un objet errors.
- Object.keys(errors).length permet de savoir s'il y a des erreurs.
- localStorage.setItem() sauvegarde les données sous forme de texte JSON.
- JSON.stringify() convertit un tableau JavaScript en chaîne JSON.
- JSON.parse() reconvertit la chaîne JSON en tableau.
- trim() supprime les espaces au début et à la fin des champs.
- includes "@" vérifie la présence d'un caractère dans une chaîne.