

Atelier 02.03 Events

Les grandes catégories d'événements en React

Catégorie	Exemple d'événements	Exemple d'usage
Souris	onClick, onDoubleClick, onMouseOver, onMouseOut, onContextMenu	Interactions bouton, hover
Clavier	onKeyDown, onKeyUp, onKeyPress	Raccourcis clavier, validation
Formulaires	onChange, onSubmit, onInput, onReset	Saisie texte, validation form
Fenêtre / Navigation	onScroll, onResize	Effets sur scroll, responsivité
Clipboard (copier/coller)	onCopy, onPaste	Restrictions sécurité, tracking
Focus / Blur	onFocus, onBlur	Mettre en évidence un champ
Médias	onPlay, onPause, onVolumeChange	Lecteurs vidéo/audio
Mobile / Touch	onTouchStart, onTouchEnd	UI mobile tactile
Événements personnalisés	Fonction callback entre composants	Communication interne React

Syntaxe : Référence vs Fonction fléchée

Passer la fonction par référence

```
<button onClick={handleClick}>Clique</button>
```

- Avantage : meilleure performance
- Utilise la fonction **déjà définie**

Passer via une fonction fléchée

```
<button onClick={() => handleClick("Kamel")}>Clique</button>
```

- Permet de **passer des arguments**
- Crée à chaque rendu

Exercice 1 — Événement onClick (base)

Objectif

Comprendre l'appel d'une fonction lors d'un clic.

Énoncé

Créer un bouton. Lorsqu'on clique dessus → afficher une alerte.



Correction

```
//src/components/ClickButton.jsx
function ClickButton() {
  function handleClick() {
    alert("Bouton cliqué !");
  }
}
```

```

return (
  <button onClick={handleClick}>
    Clique moi
  </button>
);
}

export default ClickButton;

```

Autre version

```

//src/components/ClickButton.jsx
function ClikButton(props) {

  function onClick() {
    alert(props.message || "Bouton cliqué !");
  }

  return (
    <button onClick={onClick}>
      {props.label || "Cliquez-moi"}
    </button>
  );
}
export default ClikButton;

```

```

// Fichier : src/main.jsx
import React from "react";
import ReactDOM from "react-dom/client";
import ClikButton from "./components/ClickButton.jsx";
import "./index.css";
ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <ClikButton label={"Bouton 1"} message={"Message 1"} />
    <br />
    <ClikButton label={"Bouton 2"} message={"Message 2"} />
  </React.StrictMode>
);

```

Autre version

```

//src/components/ClickButton.jsx
function ClikButton({label, message}) {

  function onClick() {
    alert(message || "Bouton cliqué !");
  }

  return (
    <button onClick={onClick}>
      {label || "Cliquez-moi"}
    </button>
  );
}
export default ClikButton;

```

Exercice 2 — onClick avec paramètre

Objectif

Utiliser une fonction fléchée pour envoyer une valeur.

Énoncé

Afficher une alerte : "Bonjour Kamel".

Correction

```
//src/components/SayHello.jsx
function SayHello() {
  function greet(name) {
    alert("Bonjour " + name);
  }

  return (
    <button onClick={() => greet("Kamel")}>
      Dire Bonjour
    </button>
  );
}

export default SayHello;
```

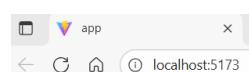
Exercice 3 — onChange : capturer la saisie

Objectif

Utiliser `event.target.value`

Énoncé

Afficher en temps réel ce que l'utilisateur tape.



Live Input Component

je tape un texte

Correction

```
// Fichier : src/components/LiveInput.jsx
import { useState } from "react";
function LiveInput() {

  const [inputValue, setInputValue] = useState("");

  function handleChange(event) {
    /* console.log("Input changed to:", event.target.value); */
    setInputValue(event.target.value);
  }

  return (
    <div>
```

```

<h2>Live Input Component</h2>
<input onChange={handleChange} type="text" placeholder="Type something..." />
<p>{inputValue}</p>
</div>
);
}
export default LiveInput;

```

Exercice 4 — Formulaire + preventDefault + Output

Objectif

Empêcher le rechargement de la page.

Énoncé

Soumettre un formulaire → afficher le nom sous le formulaire.

Correction

```

// src/components/FormSubmit.jsx
import { useState } from "react";
function FormSubmit() {

  const [text, setData] = useState(null);
  function handleSubmit(event) {
    event.preventDefault();
    const value = event.target.textInput.value;
    setData(value);
    console.log("Form submitted with data:", value);
  }

  return (
    <div>
      <form onSubmit={handleSubmit}>
        <input type="text" name="textInput" placeholder="Enter text" />
        <button type="submit">Submit</button>
      </form>
      <p>{text}</p>
    </div>
  );
}
export default FormSubmit;

```

Exercice 5 — Survol + changement de style (hover)

Objectif

Modifier des styles dynamiquement.

Survole-moi 😎

Correction

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

```

function HoverColor() {
  const [color, setColor] = useState("black");

  return (
    <h2
      style={{ color }}
      onMouseOver={() => setColor("red")}
      onMouseOut={() => setColor("black")}
    >
      Survole-moi 😊
    </h2>
  );
}

export default HoverColor;

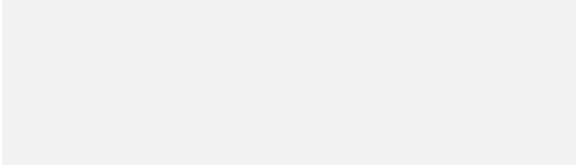
```

Exercice 6 — Déetecter la position du curseur

Objectif

Utiliser `event.clientX` & `event.clientY`.

Position : X=13 | Y=184



Créer un nouveau composant React

- Dans le dossier `src/components/`, crée un fichier nommé **MouseTracker.jsx**.

Importer le hook nécessaire

- En haut du fichier, importe le hook `useState` depuis React.

Initialiser le state

- Crée une variable `pos` qui contiendra un objet `{ x: 0, y: 0 }`.
Exemple :
- `const [pos, setPos] = useState({ x: 0, y: 0 });`

Créer une fonction qui réagit au déplacement de la souris

- Nomme-la `handleMove(e)` et récupère la position de la souris via `e.clientX` et `e.clientY`.
Mets à jour le state avec `setPos({ x: e.clientX, y: e.clientY })`.

Créer une zone de suivi

- Crée un élément `<div>` avec :
 - un événement `onMouseMove={handleMove}`
 - un style (par exemple une hauteur de 300px et un fond gris clair)
 - et affiche les coordonnées X et Y à l'intérieur.

Modifier le point d'entrée

- Ouvre src/main.jsx
- Remplace le composant affiché (ex. App, Counter, etc.) par MouseTracker :
- import MouseTracker from "./components/MouseTracker.jsx";

Correction

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

function MouseTracker() {
  const [pos, setPos] = useState({ x: 0, y: 0 });

  function handleMove(e) {
    setPos({ x: e.clientX, y: e.clientY });
  }

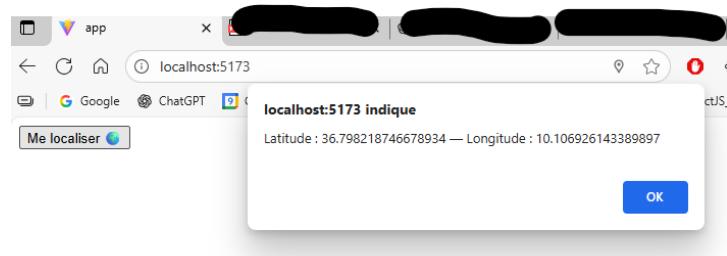
  return (
    <div
      onMouseMove={handleMove}
      style={{ height: "300px", background: "#f2f2f2" }}
    >
      Position : X={pos.x} | Y={pos.y}
    </div>
  );
}

export default MouseTracker;
```

Exercice 7 — Déterminer la position GPS

Objectif

Utiliser navigator.geolocation.



Correction

```
//src/components/GeoLocation.jsx
function GeoLocation() {
  function locate() {
    navigator.geolocation.getCurrentPosition((position) => {
      alert(`Latitude : ${position.coords.latitude} — Longitude : ${position.coords.longitude}`);
    });
  }

  return <button onClick={locate}>Me localiser </button>;
}

export default GeoLocation;
```

Exercice 8 Jeu “Clique la cible”

👉 On crée une cible qui se déplace à chaque clic.

```
//src/components/CibleGame.jsx
import { useState, useEffect } from "react";

const CibleGame = () => {
  // Etat pour position de la cible
  const [position, setPosition] = useState({ top: 100, left: 100 });
  // Etat pour score
  const [score, setScore] = useState(0);
  // Etat pour temps restant
  const [time, setTime] = useState(30);

  // Déplacement aléatoire de la cible
  const moveTarget = () => {
    const newTop = Math.floor(Math.random() * 400); // hauteur max 400px
    const newLeft = Math.floor(Math.random() * 400); // largeur max 400px
    setPosition({ top: newTop, left: newLeft });
  };

  // Gestion du clic sur la cible
  const handleClick = () => {
    setScore(score + 1);
    moveTarget();
  };

  // Timer du jeu
  useEffect(() => {
    if (time <= 0) return;
    const timer = setInterval(() => setTime(time - 1), 1000);
    return () => clearInterval(timer);
  }, [time]);

  return (
    <div
      style={{
        position: "relative",
        width: "500px",
        height: "500px",
        border: "2px solid black",
        margin: "20px auto",
        backgroundColor: "#f0f0f0",
      }}
    >
      <h2>Score: {score}</h2>
      <h3>Temps restant: {time}s</h3>
      {time > 0 && (
        <div
          onClick={handleClick} // Événement de clic
          style={{
            position: "absolute",
            top: position.top,
            left: position.left,
            width: "50px",
            height: "50px",
            backgroundColor: "red",
            borderRadius: "50%",
            cursor: "pointer",
          }}
        />
      )}
      {time === 0 && <h2>Temps écoulé ! Votre score: {score}</h2>}
    </div>
  );
};

export default CibleGame;
```