Exercice 8 - useRef et manipulation du DOM

Objectif

Utiliser useRef pour accéder aux éléments du DOM et stocker des valeurs mutables.

Concepts abordés

- useRef pour les références DOM
- useRef pour les valeurs persistantes
- Focus, scroll, mesures d'éléments
- Différence avec useState

---

useRef : Deux cas d'usage

1. Référencer un élément du DOM

```
const inputRef = useRef(null);

// Après le rendu, inputRef.current = l'élément DOM
<input ref={inputRef} />

// Accéder à l'élément
inputRef.current.focus();
inputRef.current.value;
inputRef.current.getBoundingClientRect();
```

2. Stocker une valeur mutable (sans re-render)

```
const countRef = useRef(0);

// Modifier sans déclencher de re-render
countRef.current += 1;

// Utile pour : timers, valeurs précédentes, flags
```

Énoncé

Créer plusieurs composants utilisant useRef :

1. **AutoFocusInput** : Input qui reçoit le focus automatiquement
2. **TextSelection** : Afficher le texte sélectionné dans un textarea
3. **ScrollToSection** : Navigation qui scroll vers des sections

4. **PreviousValue** : Afficher la valeur précédente d'un state

---

Correction

1. AutoFocusInput

```
import { useRef, useEffect } from 'react';

function AutoFocusInput() {
  // Créer une référence (initialement null)
  const inputRef = useRef(null);

  // Au montage, mettre le focus sur l'input
  useEffect(() => {
    // inputRef.current contient maintenant l'élément DOM
    if (inputRef.current) {
      inputRef.current.focus();
    }
  }, []);

  return (
    <div className="auto-focus-demo">
      <h3>Auto Focus</h3>
      <input
        ref={inputRef}  // Connecter la ref à l'élément
        type="text"
        placeholder="Je reçois le focus automatiquement"
      />
    </div>
  );
}
```

2. TextSelection

```
import { useRef, useState } from 'react';

function TextSelection() {
  const textareaRef = useRef(null);
  const [selection, setSelection] = useState('');

  function handleSelect() {
    const textarea = textareaRef.current;
    if (textarea) {
      const start = textarea.selectionStart;
      const end = textarea.selectionEnd;
      const selectedText = textarea.value.substring(start, end);
      setSelection(selectedText);
    }
  }
```

```
  return (
    <div className="text-selection-demo">
      <h3>Sélection de texte</h3>
      <textarea
        ref={textareaRef}
        onSelect={handleSelect}
        defaultValue="Sélectionnez une partie de ce texte pour voir ce qui est
sélectionné."
        rows={4}
        cols={50}
      />
      {selection && (
        <p>
          Texte sélectionné : <strong>"{selection}"</strong>
        </p>
      )}
    </div>
  );
}
```

3. ScrollToSection

```
import { useRef } from 'react';

function ScrollToSection() {
  // Une ref pour chaque section
  const section1Ref = useRef(null);
  const section2Ref = useRef(null);
  const section3Ref = useRef(null);

  function scrollTo(ref) {
    ref.current?.scrollIntoView({
      behavior: 'smooth',
      block: 'start'
    });
  }

  return (
    <div className="scroll-demo">
      {/* Navigation fixe */}
      <nav className="scroll-nav">
        <button onClick={() => scrollTo(section1Ref)}>Section 1</button>
        <button onClick={() => scrollTo(section2Ref)}>Section 2</button>
        <button onClick={() => scrollTo(section3Ref)}>Section 3</button>
      </nav>

      {/* Sections */}
      <section ref={section1Ref} className="scroll-section">
        <h2>Section 1</h2>
        <p>Contenu de la section 1...</p>
```

```
      </section>

      <section ref={section2Ref} className="scroll-section">
        <h2>Section 2</h2>
        <p>Contenu de la section 2...</p>
      </section>

      <section ref={section3Ref} className="scroll-section">
        <h2>Section 3</h2>
        <p>Contenu de la section 3...</p>
      </section>
    </div>
  );
}
```

4. PreviousValue (useRef pour valeur mutable)

```
import { useState, useRef, useEffect } from 'react';

// Hook personnalisé pour obtenir la valeur précédente
function usePrevious(value) {
  const ref = useRef();

  useEffect(() => {
    ref.current = value;
  }, [value]);

  return ref.current;
}

function PreviousValueDemo() {
  const [count, setCount] = useState(0);
  const previousCount = usePrevious(count);

  return (
    <div className="previous-value-demo">
      <h3>Valeur précédente</h3>
      <p>Actuel : {count}</p>
      <p>Précédent : {previousCount ?? 'N/A'}</p>
      <button onClick={() => setCount(c => c + 1)}>+1</button>
      <button onClick={() => setCount(c => c - 1)}>-1</button>
    </div>
  );
}
```

5. Mesurer un élément

```
import { useRef, useState, useEffect } from 'react';
```

```jsx
function MeasureElement() {
  const boxRef = useRef(null);
  const [dimensions, setDimensions] = useState({ width: 0, height: 0 });

  useEffect(() => {
    function updateDimensions() {
      if (boxRef.current) {
        const rect = boxRef.current.getBoundingClientRect();
        setDimensions({
          width: Math.round(rect.width),
          height: Math.round(rect.height)
        });
      }
    }

    updateDimensions();
    window.addEventListener('resize', updateDimensions);

    return () => window.removeEventListener('resize', updateDimensions);
  }, []);

  return (
    <div className="measure-demo">
      <h3>Mesurer un élément</h3>
      <div
        ref={boxRef}
        className="measured-box"
        style={{ width: '50%', height: '100px', background: '#3b82f6' }}
      >
        Redimensionnez la fenêtre
      </div>
      <p>
        Dimensions : {dimensions.width}px × {dimensions.height}px
      </p>
    </div>
  );
}
```

CSS

```css
/* useRef-demo.css */
.scroll-nav {
  position: sticky;
  top: 0;
  background: white;
  padding: 1rem;
  display: flex;
  gap: 0.5rem;
  border-bottom: 1px solid #e5e7eb;
}
```

```css
.scroll-section {
  min-height: 300px;
  padding: 2rem;
  border: 1px solid #e5e7eb;
  margin: 1rem 0;
}

.measured-box {
  display: flex;
  align-items: center;
  justify-content: center;
  color: white;
  border-radius: 8px;
}
```

---

useState vs useRef

Caractéristique	useState	useRef
Déclenche un re-render	☑ Oui	❌ Non
Valeur disponible après render	☑ Oui	☑ Oui
Accès à l'élément DOM	❌ Non	☑ Oui
Persiste entre les renders	☑ Oui	☑ Oui

Utilisez useRef quand :

- Vous devez accéder au DOM
- Vous voulez stocker une valeur sans re-render (timer ID, valeur précédente)