Exercice 3 - Timer avec setInterval

Objectif

Créer un timer qui se met à jour chaque seconde en gérant correctement le cleanup.

Concepts abordés

- useEffect avec cleanup
- setInterval et clearInterval
- useRef pour valeurs persistantes
- Gestion du cycle de vie

Énoncé

Créer un composant `Timer` qui :

1. Affiche un compteur qui s'incrémente chaque seconde
2. A un bouton Play/Pause
3. A un bouton Reset
4. Nettoie l'intervalle quand le composant est démonté

Indices

▶ 💡 setInterval dans useEffect

```
useEffect(() => {
  const intervalId = setInterval(() => {
    // Code exécuté toutes les X ms
  }, 1000);

  // Cleanup : nettoyer l'intervalle
  return () => clearInterval(intervalId);
}, []);
```

▶ 💡 Problème du state "stale" (périmé)

```
// ❌ PROBLÈME : count garde sa valeur initiale dans le closure
useEffect(() => {
  const id = setInterval(() => {
    setCount(count + 1);  // count = toujours 0 !
  }, 1000);
  return () => clearInterval(id);
}, []);
```

```
// ☑ SOLUTION : utiliser la forme fonction de setState
useEffect(() => {
  const id = setInterval(() => {
    setCount(prev => prev + 1);  // OK !
  }, 1000);
  return () => clearInterval(id);
}, []);
```

Correction

```
import { useState, useEffect, useRef } from 'react';
import './Timer.css';

function Timer() {
  const [seconds, setSeconds] = useState(0);
  const [isRunning, setIsRunning] = useState(false);

  // useRef pour stocker l'ID de l'intervalle
  // (ne déclenche pas de re-render quand il change)
  const intervalRef = useRef(null);

  // Effect pour gérer l'intervalle
  useEffect(() => {
    if (isRunning) {
      // Démarrer l'intervalle
      intervalRef.current = setInterval(() => {
        setSeconds(prev => prev + 1);
      }, 1000);
    }

    // Cleanup : nettoyer quand isRunning change ou démontage
    return () => {
      if (intervalRef.current) {
        clearInterval(intervalRef.current);
        intervalRef.current = null;
      }
    };
  }, [isRunning]);

  function handlePlayPause() {
    setIsRunning(prev => !prev);
  }

  function handleReset() {
    setIsRunning(false);
    setSeconds(0);
  }

  // Formatage du temps
  const minutes = Math.floor(seconds / 60);
```

```
  const secs = seconds % 60;
  const display = `${minutes.toString().padStart(2,
'0')}:${secs.toString().padStart(2, '0')}`;

  return (
    <div className="timer">
      <div className="timer-display">{display}</div>
      <div className="timer-controls">
        <button onClick={handlePlayPause} className="control-btn">
          {isRunning ? '⏸ Pause' : '▶ Play'}
        </button>
        <button onClick={handleReset} className="control-btn reset">
          🔄 Reset
        </button>
      </div>
    </div>
  );
}

export default Timer;
```

```
/* Timer.css */
.timer {
  text-align: center;
  padding: 2rem;
  background: white;
  border-radius: 12px;
  box-shadow: 0 2px 8px rgba(0,0,0,0.1);
  max-width: 300px;
  margin: 0 auto;
}

.timer-display {
  font-size: 4rem;
  font-family: monospace;
  color: #1f2937;
  margin-bottom: 1.5rem;
}

.timer-controls {
  display: flex;
  gap: 1rem;
  justify-content: center;
}

.control-btn {
  padding: 0.75rem 1.5rem;
  font-size: 1rem;
  border: none;
  border-radius: 8px;
  cursor: pointer;
```

```
  background: #3b82f6;
  color: white;
  transition: background 0.2s;
}

.control-btn:hover {
  background: #2563eb;
}

.control-btn.reset {
  background: #6b7280;
}

.control-btn.reset:hover {
  background: #4b5563;
}
```

Points d'attention

⚠ Pourquoi useRef pour l'intervalId ?

- `useState` : déclenche un re-render à chaque changement
- `useRef` : persiste la valeur SANS re-render

L'ID de l'intervalle n'a pas besoin d'être affiché, donc useRef est plus approprié.

⚠ Le cleanup est CRUCIAL

Sans cleanup, l'intervalle continue même après démontage du composant, causant :

- Memory leaks
- Erreurs "Can't perform state update on unmounted component"