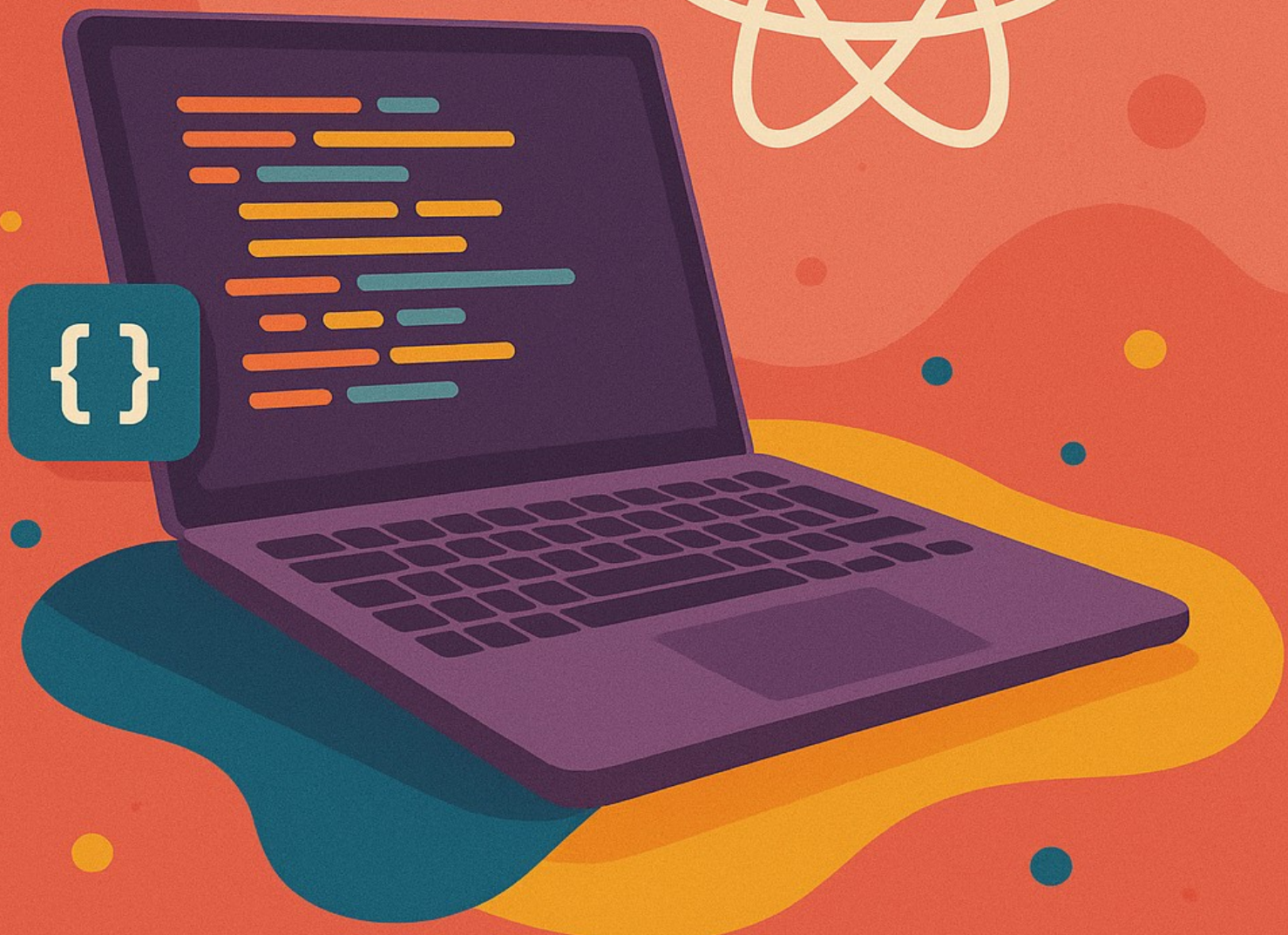
EXERCICES ET CORRIGÉS REACT JS



React JS

10 Exercices Corrigés

Du Niveau Débutant au Niveau Avancé

Partie 2 : Exercices 4-7 (Niveau Intermédiaire)

Exercice 4 : Formulaire de Connexion

⭐⭐⭐ **Niveau : Intermédiaire**

📋 Énoncé

📝 **Créez un formulaire de connexion avec email et mot de passe. Le formulaire doit valider les champs en temps réel et afficher des messages d'erreur appropriés. Le bouton de soumission doit être désactivé tant que le formulaire contient des erreurs.**

🎯 Objectifs d'apprentissage

- Gérer un formulaire avec plusieurs champs
- Implémenter une validation en temps réel
- Afficher des messages d'erreur conditionnels
- Désactiver/activer des boutons selon l'état

📝 Spécifications

- Email doit être au format valide (contenir @)
- Mot de passe minimum 6 caractères
- Messages d'erreur en rouge sous chaque champ
- Message de succès après soumission valide

💡 *Conseil : Utilisez deux states séparés - un pour les valeurs du formulaire, un pour les erreurs.*

💭 Indices

- **Indice 1 :** Utilisez une regex pour valider l'email : /^[^\s@]+@[^\s@]+\.[^\s@]+$/
- **Indice 2 :** Pour gérer plusieurs champs : const { name, value } = e.target
- **Indice 3 :** Désactiver le bouton : disabled={Object.keys(erreurs).length > 0}

✅ Solution Complète

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

function FormulaireConnexion() {
  // États du formulaire
  const [form, setForm] = useState({
    email: '',
    password: ''
  });

  const [erreurs, setErreurs] = useState({});
  const [soumis, setSoumis] = useState(false);

  // Fonction de validation email
  const validerEmail = (email) => {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
  };

  // Gérer les changements dans les champs
  const handleChange = (e) => {
    const { name, value } = e.target;

    // Mettre à jour la valeur
    setForm({ ...form, [name]: value });

    // Validation en temps réel
    const nouvellesErreurs = { ...erreurs };

    if (name === 'email') {
      if (!validerEmail(value)) {
        nouvellesErreurs.email = 'Format email invalide';
      } else {
```

```
        delete nouvellesErreurs.email;
      }
    }

    if (name === 'password') {
      if (value.length < 6) {
        nouvellesErreurs.password = 'Minimum 6 caractères requis';
      } else {
        delete nouvellesErreurs.password;
      }
    }

    setErreurs(nouvellesErreurs);
  };

  // Gérer la soumission
  const handleSubmit = (e) => {
    e.preventDefault();

    if (Object.keys(erreurs).length === 0 && form.email && form.password) {
      setSoumis(true);
      console.log('Formulaire soumis:', form);

      // Réinitialiser après 3 secondes
      setTimeout(() => {
        setSoumis(false);
        setForm({ email: '', password: '' });
      }, 3000);
    }
  };

  return (
    <div style={{
      maxWidth: '450px',
      margin: '50px auto',
      padding: '40px',
      backgroundColor: 'white',
      borderRadius: '15px',
      boxShadow: '0 4px 20px rgba(0, 0, 0, 0.1)',
      fontFamily: 'Arial, sans-serif'
    }}>
      <h2 style={{
        textAlign: 'center',
```

```
        color: '#2c3e50',
        marginBottom: '30px',
        fontSize: '2rem'
      }}>
        🔐 Connexion
      </h2>

      {/* Message de succès */}
      {soumis && (
        <div style={{
          padding: '15px',
          backgroundColor: '#d4edda',
          color: '#155724',
          borderRadius: '8px',
          marginBottom: '25px',
          textAlign: 'center',
          fontWeight: 'bold'
        }}>
          ✅ Connexion réussie !
        </div>
      )}

      <form onSubmit={handleSubmit}>
        {/* Champ Email */}
        <div style={{ marginBottom: '25px' }}>
          <label style={{
            display: 'block',
            marginBottom: '8px',
            color: '#2c3e50',
            fontWeight: 'bold'
          }}>
            Email :
          </label>
          <input
            type="email"
            name="email"
            value={form.email}
            onChange={handleChange}
            placeholder="votre@email.com"
            style={{
              width: '100%',
              padding: '12px',
              border: erreurs.email ? '2px solid #e74c3c' : '2px solid #ddd',
              borderRadius: '8px',
```

```
            fontSize: '1rem',
            outline: 'none',
            transition: 'border-color 0.3s'
          }}
        />
        {erreurs.email && (
          <p style={{
            color: '#e74c3c',
            fontSize: '0.9rem',
            marginTop: '5px',
            marginBottom: 0
          }}>
            ⚠️ {erreurs.email}
          </p>
        )}
      </div>

      {/* Champ Mot de passe */}
      <div style={{ marginBottom: '25px' }}>
        <label style={{
          display: 'block',
          marginBottom: '8px',
          color: '#2c3e50',
          fontWeight: 'bold'
        }}>
          Mot de passe :
        </label>
        <input
          type="password"
          name="password"
          value={form.password}
          onChange={handleChange}
          placeholder="••••••"
          style={{
            width: '100%',
            padding: '12px',
            border: erreurs.password ? '2px solid #e74c3c' : '2px solid #ddd',
            borderRadius: '8px',
            fontSize: '1rem',
            outline: 'none',
            transition: 'border-color 0.3s'
          }}
        />
        {erreurs.password && (
```

```
            <p style={{
              color: '#e74c3c',
              fontSize: '0.9rem',
              marginTop: '5px',
              marginBottom: 0
            }}>
              ⚠️ {erreurs.password}
            </p>
          )}
        </div>

        {/* Bouton de soumission */}
        <button
          type="submit"
          disabled={Object.keys(erreurs).length > 0 || !form.email || !
form.password}
          style={{
            width: '100%',
            padding: '14px',
            backgroundColor: Object.keys(erreurs).length > 0 || !form.email || !
form.password
              ? '#bdc3c7'
              : '#3498db',
            color: 'white',
            border: 'none',
            borderRadius: '8px',
            fontSize: '1.1rem',
            fontWeight: 'bold',
            cursor: Object.keys(erreurs).length > 0 || !form.email || !
form.password
              ? 'not-allowed'
              : 'pointer',
            transition: 'background-color 0.3s'
          }}
        >
          Se connecter
        </button>
      </form>
    </div>
  );
}

export default FormulaireConnexion;
```

📖 Explications Détaillées

1. États multiples

```
const [form, setForm] = useState({ email: '', password: '' });
const [erreurs, setErreurs] = useState({});
const [soumis, setSoumis] = useState(false);
```

form contient les valeurs des champs, **erreurs** stocke les messages d'erreur, **soumis** indique si le formulaire a été soumis avec succès.

2. Validation d'email avec regex

```
const validerEmail = (email) => {
  const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
};
```

Cette regex vérifie que l'email contient un @ et un point, sans espaces.

3. Gestion dynamique des champs

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

En utilisant `[name]`, on peut gérer plusieurs champs avec une seule fonction. Le nom de l'attribut `name` devient la clé dans l'objet.

4. Gestion des erreurs

```
if (!validerEmail(value)) {
  nouvellesErreurs.email = 'Format email invalide';
} else {
  delete nouvellesErreurs.email;
}
```

On ajoute une erreur si la validation échoue, et on la supprime avec **delete** si elle réussit.

> **✅ Résultat : Un formulaire robuste avec validation en temps réel et retour utilisateur clair !**

Exercice 5 : Recherche de Produits

⭐⭐⭐ **Niveau : Intermédiaire**

📋 Énoncé

📝 Créez une application qui affiche une liste de produits avec recherche en temps réel et filtres par catégorie. L'utilisateur doit pouvoir rechercher par nom et filtrer par catégorie simultanément.

🎯 Objectifs d'apprentissage

- Filtrer des tableaux avec plusieurs critères
- Combiner recherche et filtres
- Afficher des données dynamiques
- Gérer un tableau d'objets complexes

📝 Données de test

```
const produits = [
  { id: 1, nom: 'iPhone 15', categorie: 'Téléphones', prix: 999 },
  { id: 2, nom: 'MacBook Pro', categorie: 'Ordinateurs', prix: 2499 },
  { id: 3, nom: 'iPad Air', categorie: 'Tablettes', prix: 599 },
  { id: 4, nom: 'Samsung Galaxy', categorie: 'Téléphones', prix: 899 },
  { id: 5, nom: 'Dell XPS', categorie: 'Ordinateurs', prix: 1799 },
  { id: 6, nom: 'Surface Pro', categorie: 'Tablettes', prix: 1099 }
];
```

💡 Conseil : Utilisez .filter() deux fois - une pour la recherche, une pour la catégorie.

✅ Solution Complète

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

function RechercheProduits() {
  const produitsData = [
    { id: 1, nom: 'iPhone 15', categorie: 'Téléphones', prix: 999 },
    { id: 2, nom: 'MacBook Pro', categorie: 'Ordinateurs', prix: 2499 },
    { id: 3, nom: 'iPad Air', categorie: 'Tablettes', prix: 599 },
    { id: 4, nom: 'Samsung Galaxy', categorie: 'Téléphones', prix: 899 },
    { id: 5, nom: 'Dell XPS', categorie: 'Ordinateurs', prix: 1799 },
    { id: 6, nom: 'Surface Pro', categorie: 'Tablettes', prix: 1099 },
    { id: 7, nom: 'AirPods Pro', categorie: 'Audio', prix: 249 },
    { id: 8, nom: 'Sony WH-1000XM5', categorie: 'Audio', prix: 399 }
  ];

  const [recherche, setRecherche] = useState('');
  const [categorieSelectionnee, setCategorieSelectionnee] = useState('Toutes');

  // Obtenir les catégories uniques
  const categories = ['Toutes', ...new Set(produitsData.map(p => p.categorie))];

  // Filtrer les produits
  const produitsFiltres = produitsData.filter(produit => {
    // Filtre par recherche (nom)
    const matchRecherche =
produit.nom.toLowerCase().includes(recherche.toLowerCase());

    // Filtre par catégorie
    const matchCategorie = categorieSelectionnee === 'Toutes' ||
                           produit.categorie === categorieSelectionnee;

    return matchRecherche && matchCategorie;
  });

  return (
    <div style={{
      maxWidth: '900px',
      margin: '50px auto',
      padding: '30px',
      fontFamily: 'Arial, sans-serif'
    }}>
      <h1 style={{
        textAlign: 'center',
```

```
        color: '#2c3e50',
        marginBottom: '40px'
      }}>
        🛍️ Recherche de Produits
      </h1>

      {/* Zone de filtres */}
      <div style={{
        display: 'flex',
        gap: '15px',
        marginBottom: '30px',
        flexWrap: 'wrap'
      }}>
        {/* Barre de recherche */}
        <input
          type="text"
          value={recherche}
          onChange={(e) => setRecherche(e.target.value)}
          placeholder="🔍 Rechercher un produit..."
          style={{
            flex: 1,
            minWidth: '250px',
            padding: '12px 15px',
            border: '2px solid #3498db',
            borderRadius: '8px',
            fontSize: '1rem',
            outline: 'none'
          }}
        />

        {/* Sélecteur de catégorie */}
        <select
          value={categorieSelectionnee}
          onChange={(e) => setCategorieSelectionnee(e.target.value)}
          style={{
            padding: '12px 15px',
            border: '2px solid #3498db',
            borderRadius: '8px',
            fontSize: '1rem',
            cursor: 'pointer',
            backgroundColor: 'white',
            minWidth: '180px'
          }}
        >
```

```
          {categories.map(cat => (
            <option key={cat} value={cat}>{cat}</option>
          ))}
        </select>
      </div>

      {/* Compteur de résultats */}
      <p style={{
        textAlign: 'center',
        color: '#7f8c8d',
        fontSize: '1.1rem',
        marginBottom: '25px'
      }}>
        {produitsFiltres.length} produit(s) trouvé(s)
      </p>

      {/* Grille de produits */}
      {produitsFiltres.length === 0 ? (
        <div style={{
          textAlign: 'center',
          padding: '60px 20px',
          color: '#95a5a6',
          fontSize: '1.2rem'
        }}>
          Aucun produit ne correspond à votre recherche 🔍
        </div>
      ) : (
        <div style={{
          display: 'grid',
          gridTemplateColumns: 'repeat(auto-fill, minmax(250px, 1fr))',
          gap: '20px'
        }}>
          {produitsFiltres.map(produit => (
            <div
              key={produit.id}
              style={{
                padding: '20px',
                backgroundColor: 'white',
                borderRadius: '12px',
                boxShadow: '0 2px 8px rgba(0, 0, 0, 0.1)',
                transition: 'transform 0.2s, box-shadow 0.2s',
                cursor: 'pointer'
              }}
              onMouseEnter={(e) => {
```

```
                e.currentTarget.style.transform = 'translateY(-5px)';
                e.currentTarget.style.boxShadow = '0 4px 16px rgba(0, 0, 0, 0.15)';
              }}
              onMouseLeave={(e) => {
                e.currentTarget.style.transform = 'translateY(0)';
                e.currentTarget.style.boxShadow = '0 2px 8px rgba(0, 0, 0, 0.1)';
              }}
            >
              <h3 style={{
                color: '#2c3e50',
                marginBottom: '10px',
                fontSize: '1.3rem'
              }}>
                {produit.nom}
              </h3>

              <span style={{
                display: 'inline-block',
                padding: '5px 12px',
                backgroundColor: '#3498db',
                color: 'white',
                borderRadius: '15px',
                fontSize: '0.85rem',
                marginBottom: '15px'
              }}>
                {produit.categorie}
              </span>

              <p style={{
                fontSize: '1.5rem',
                color: '#27ae60',
                fontWeight: 'bold',
                margin: 0
              }}>
                {produit.prix}€
              </p>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
```

```
export default RechercheProduits;
```

📖 Explications Détaillées

1. Obtenir les catégories uniques

```
const categories = ['Toutes', ...new Set(produitsData.map(p => p.categorie))];
```

new Set() élimine les doublons. **map()** extrait toutes les catégories, puis Set ne garde que les valeurs uniques.

2. Filtrage multiple

```
const produitsFiltres = produitsData.filter(produit => {
  const matchRecherche =
produit.nom.toLowerCase().includes(recherche.toLowerCase());
  const matchCategorie = categorieSelectionnee === 'Toutes' ||
                         produit.categorie === categorieSelectionnee;
  return matchRecherche && matchCategorie;
});
```

Les deux conditions (recherche ET catégorie) doivent être vraies avec l'opérateur **&&**. Si la catégorie est 'Toutes', on accepte tous les produits.

3. Grid CSS responsive

```
gridTemplateColumns: 'repeat(auto-fill, minmax(250px, 1fr))'
```

Cette propriété CSS crée une grille qui s'adapte automatiquement : au moins 250px par colonne, avec autant de colonnes que possible.

> **✅ Résultat : Une interface de recherche fluide et réactive avec filtres combinés !**

Exercice 6 : Chronomètre

⭐⭐⭐ **Niveau : Intermédiaire+**

📋 Énoncé

📝 **Créez un chronomètre avec les fonctionnalités : Démarrer, Pause, Arrêter, et enregistrer des tours. Le temps doit s'afficher au format MM:SS:MS (minutes, secondes, millisecondes).**

🎯 Objectifs d'apprentissage

- Utiliser useEffect avec setInterval
- Nettoyer les effets (cleanup)
- Formater du temps
- Gérer plusieurs états interdépendants

💡 *Conseil : Stockez le temps en millisecondes, puis convertissez pour l'affichage.*

✅ Solution Complète

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

function Chronometre() {
  const [temps, setTemps] = useState(0);
  const [enMarche, setEnMarche] = useState(false);
  const [tours, setTours] = useState([]);

  // useEffect pour le chronomètre
  useEffect(() => {
    let intervalle;

    if (enMarche) {
      intervalle = setInterval(() => {
        setTemps(prevTemps => prevTemps + 10);
      }, 10);
    }

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

  // Formater le temps
  const formaterTemps = (ms) => {
    const minutes = Math.floor(ms / 60000);
    const secondes = Math.floor((ms % 60000) / 1000);
    const millisecondes = Math.floor((ms % 1000) / 10);

    return `${minutes.toString().padStart(2, '0')}:$
{secondes.toString().padStart(2, '0')}:${millisecondes.toString().padStart(2,
'0')}`;
  };

  // Fonctions de contrôle
  const demarrer = () => setEnMarche(true);
  const pause = () => setEnMarche(false);

  const arreter = () => {
    setEnMarche(false);
    setTemps(0);
    setTours([]);
  };
```

```
const enregistrerTour = () => {
  setTours([...tours, temps]);
};

return (
  <div style={{
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    justifyContent: 'center',
    minHeight: '100vh',
    backgroundColor: '#2c3e50',
    color: 'white',
    fontFamily: 'Arial, sans-serif'
  }}>
    <h1 style={{ marginBottom: '40px', fontSize: '2.5rem' }}>
      ⏱️ Chronomètre
    </h1>

    {/* Affichage du temps */}
    <div style={{
      fontSize: '4rem',
      fontWeight: 'bold',
      marginBottom: '40px',
      fontFamily: 'monospace',
      backgroundColor: 'rgba(255, 255, 255, 0.1)',
      padding: '30px 50px',
      borderRadius: '15px',
      minWidth: '350px',
      textAlign: 'center'
    }}>
      {formaterTemps(temps)}
    </div>

    {/* Boutons de contrôle */}
    <div style={{
      display: 'flex',
      gap: '15px',
      marginBottom: '40px',
      flexWrap: 'wrap',
      justifyContent: 'center'
    }}>
      {!enMarche ? (
```

```
          <button
            onClick={demarrer}
            style={{
              padding: '15px 35px',
              fontSize: '1.2rem',
              fontWeight: 'bold',
              backgroundColor: '#27ae60',
              color: 'white',
              border: 'none',
              borderRadius: '10px',
              cursor: 'pointer'
            }}
          >
            ▶️ Démarrer
          </button>
        ) : (
          <button
            onClick={pause}
            style={{
              padding: '15px 35px',
              fontSize: '1.2rem',
              fontWeight: 'bold',
              backgroundColor: '#f39c12',
              color: 'white',
              border: 'none',
              borderRadius: '10px',
              cursor: 'pointer'
            }}
          >
            ⏸️ Pause
          </button>
        )}

        <button
          onClick={arreter}
          disabled={temps === 0}
          style={{
            padding: '15px 35px',
            fontSize: '1.2rem',
            fontWeight: 'bold',
            backgroundColor: temps === 0 ? '#7f8c8d' : '#e74c3c',
            color: 'white',
            border: 'none',
            borderRadius: '10px',
```

```
            cursor: temps === 0 ? 'not-allowed' : 'pointer'
          }}
        >
          ⏹️ Arrêter
        </button>

        <button
          onClick={enregistrerTour}
          disabled={temps === 0 || !enMarche}
          style={{
            padding: '15px 35px',
            fontSize: '1.2rem',
            fontWeight: 'bold',
            backgroundColor: temps === 0 || !enMarche ? '#7f8c8d' : '#3498db',
            color: 'white',
            border: 'none',
            borderRadius: '10px',
            cursor: temps === 0 || !enMarche ? 'not-allowed' : 'pointer'
          }}
        >
          🏁 Tour
        </button>
      </div>

      {/* Liste des tours */}
      {tours.length > 0 && (
        <div style={{
          backgroundColor: 'rgba(255, 255, 255, 0.1)',
          borderRadius: '15px',
          padding: '25px',
          minWidth: '350px',
          maxHeight: '300px',
          overflowY: 'auto'
        }}>
          <h3 style={{ marginBottom: '20px', textAlign: 'center' }}>
            📊 Tours enregistrés
          </h3>
          {tours.map((tour, index) => (
            <div
              key={index}
              style={{
                display: 'flex',
                justifyContent: 'space-between',
                padding: '12px 20px',
```

```
                marginBottom: '10px',
                backgroundColor: 'rgba(255, 255, 255, 0.05)',
                borderRadius: '8px',
                borderLeft: '4px solid #3498db'
              }}
            >
              <span style={{ fontWeight: 'bold' }}>
                Tour {index + 1}
              </span>
              <span style={{ fontFamily: 'monospace', fontSize: '1.1rem' }}>
                {formaterTemps(tour)}
              </span>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}

export default Chronometre;
```

📖 Explications Détaillées

1. useEffect avec setInterval

```
useEffect(() => {
  let intervalle;
  if (enMarche) {
    intervalle = setInterval(() => {
      setTemps(prevTemps => prevTemps + 10);
    }, 10);
  }
  return () => clearInterval(intervalle);
}, [enMarche]);
```

L'effet s'exécute quand **enMarche** change. Si true, on démarre un intervalle qui ajoute 10ms toutes les 10 millisecondes.

La fonction return nettoie l'intervalle quand le composant se démonte ou quand enMarche change.

2. Formatage du temps

```
const formaterTemps = (ms) => {
  const minutes = Math.floor(ms / 60000);
  const secondes = Math.floor((ms % 60000) / 1000);
  const millisecondes = Math.floor((ms % 1000) / 10);

  return `${minutes.toString().padStart(2, '0')}:${secondes.toString().padStart(2,
'0')}:${millisecondes.toString().padStart(2, '0')}`;
};
```

`padStart(2, '0')` ajoute un zéro devant si le nombre n'a qu'un chiffre (ex: 5 devient 05).

`ms % 60000` donne le reste de la division par 60000, c'est-à-dire les millisecondes qui ne font pas une minute complète.

3. Enregistrement des tours

```
const enregistrerTour = () => {
  setTours([...tours, temps]);
};
```

On ajoute le temps actuel à la fin du tableau tours. Chaque tour est stocké indépendamment.

> **✅ Résultat : Un chronomètre professionnel avec gestion précise du temps et des tours !**

Exercice 7 : Quiz Interactif

⭐⭐⭐⭐ **Niveau : Intermédiaire+**

📋 Énoncé

> **📝 Créez un quiz avec plusieurs questions à choix multiples. L'utilisateur répond aux questions une par une, et à la fin voit son score total avec un récapitulatif de ses réponses.**

🎯 Objectifs d'apprentissage

- Gérer une navigation entre étapes
- Calculer un score
- Afficher différentes vues conditionnellement
- Manipuler un tableau d'objets

📝 Données du quiz

```
const questions = [
  {
    id: 1,
    question: "Quel hook permet de gérer l'état dans React ?",
    options: ["useEffect", "useState", "useContext", "useRef"],
    reponseCorrecte: "useState"
  },
  {
    id: 2,
    question: "Que signifie JSX ?",
    options: ["Java Syntax Extension", "JavaScript XML", "JSON Extension", "Java
Source XML"],
    reponseCorrecte: "JavaScript XML"
  },
  // ... autres questions
];
```

✅ Solution Complète

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

function Quiz() {
  const questions = [
    {
      id: 1,
      question: "Quel hook permet de gérer l'état dans React ?",
      options: ["useEffect", "useState", "useContext", "useRef"],
      reponseCorrecte: "useState"
    },
    {
      id: 2,
      question: "Que signifie JSX ?",
      options: ["Java Syntax Extension", "JavaScript XML", "JSON Extension", "Java
Source XML"],
      reponseCorrecte: "JavaScript XML"
    },
    {
      id: 3,
      question: "Quelle méthode affiche une liste en React ?",
      options: ["forEach()", "map()", "filter()", "reduce()"],
      reponseCorrecte: "map()"
    },
    {
      id: 4,
      question: "Comment passe-t-on des données à un composant enfant ?",
      options: ["Via state", "Via props", "Via context", "Via refs"],
      reponseCorrecte: "Via props"
    },
    {
      id: 5,
      question: "Quel hook gère les effets de bord ?",
      options: ["useState", "useEffect", "useCallback", "useMemo"],
      reponseCorrecte: "useEffect"
    }
  ];

  const [questionActuelle, setQuestionActuelle] = useState(0);
  const [reponses, setReponses] = useState([]);
  const [termine, setTermine] = useState(false);

  const handleReponse = (option) => {
    const nouvellesReponses = [...reponses, {
      question: questions[questionActuelle].question,
```

```
      reponseUtilisateur: option,
      reponseCorrecte: questions[questionActuelle].reponseCorrecte,
      correct: option === questions[questionActuelle].reponseCorrecte
    }];

    setReponses(nouvellesReponses);

    if (questionActuelle + 1 < questions.length) {
      setQuestionActuelle(questionActuelle + 1);
    } else {
      setTermine(true);
    }
  };

  const recommencer = () => {
    setQuestionActuelle(0);
    setReponses([]);
    setTermine(false);
  };

  const calculerScore = () => {
    return reponses.filter(r => r.correct).length;
  };

  // Écran de résultats
  if (termine) {
    const score = calculerScore();
    const pourcentage = Math.round((score / questions.length) * 100);

    return (
      <div style={{
        maxWidth: '700px',
        margin: '50px auto',
        padding: '40px',
        backgroundColor: 'white',
        borderRadius: '15px',
        boxShadow: '0 4px 20px rgba(0, 0, 0, 0.1)',
        fontFamily: 'Arial, sans-serif'
      }}>
        <h1 style={{ textAlign: 'center', color: '#2c3e50', marginBottom:
'30px' }}>
          🎉 Quiz Terminé !
        </h1>
```

```
        {/* Score */}
        <div style={{
          textAlign: 'center',
          padding: '30px',
          backgroundColor: pourcentage >= 70 ? '#d4edda' : '#f8d7da',
          borderRadius: '10px',
          marginBottom: '30px'
        }}>
          <h2 style={{
            fontSize: '3rem',
            margin: 0,
            color: pourcentage >= 70 ? '#27ae60' : '#e74c3c'
          }}>
            {score} / {questions.length}
          </h2>
          <p style={{ fontSize: '1.5rem', margin: '10px 0' }}>
            {pourcentage}%
          </p>
          <p style={{ fontSize: '1.2rem', fontWeight: 'bold' }}>
            {pourcentage >= 90 ? '🏆 Excellent !' :
             pourcentage >= 70 ? '👍 Bien joué !' :
             pourcentage >= 50 ? '😊 Pas mal !' :
             '💪 Continue à apprendre !'}
          </p>
        </div>

        {/* Récapitulatif */}
        <h3 style={{ color: '#2c3e50', marginBottom: '20px' }}>
          📋 Récapitulatif des réponses
        </h3>
        {reponses.map((rep, index) => (
          <div
            key={index}
            style={{
              padding: '20px',
              marginBottom: '15px',
              backgroundColor: rep.correct ? '#d4edda' : '#f8d7da',
              borderRadius: '10px',
              borderLeft: `5px solid ${rep.correct ? '#27ae60' : '#e74c3c'}`
            }}
          >
            <p style={{ fontWeight: 'bold', marginBottom: '10px', color:
'#2c3e50' }}>
              Question {index + 1}: {rep.question}
```

```
            </p>
            <p style={{ margin: '5px 0' }}>
              {rep.correct ? '✅' : '❌'} Votre réponse :
<strong>{rep.reponseUtilisateur}</strong>
            </p>
            {!rep.correct && (
              <p style={{ margin: '5px 0', color: '#27ae60' }}>
                ✓ Bonne réponse : <strong>{rep.reponseCorrecte}</strong>
              </p>
            )}
          </div>
        ))}

        <button
          onClick={recommencer}
          style={{
            width: '100%',
            padding: '15px',
            backgroundColor: '#3498db',
            color: 'white',
            border: 'none',
            borderRadius: '10px',
            fontSize: '1.2rem',
            fontWeight: 'bold',
            cursor: 'pointer',
            marginTop: '20px'
          }}
        >
          🔄 Recommencer le quiz
        </button>
      </div>
    );
  }

  // Écran de question
  const question = questions[questionActuelle];

  return (
    <div style={{
      maxWidth: '700px',
      margin: '50px auto',
      padding: '40px',
      backgroundColor: 'white',
      borderRadius: '15px',
```

```
      boxShadow: '0 4px 20px rgba(0, 0, 0, 0.1)',
      fontFamily: 'Arial, sans-serif'
    }}>
      {/* En-tête */}
      <div style={{
        display: 'flex',
        justifyContent: 'space-between',
        alignItems: 'center',
        marginBottom: '30px',
        paddingBottom: '20px',
        borderBottom: '2px solid #ecf0f1'
      }}>
        <h2 style={{ color: '#3498db', margin: 0 }}>
          📝 Quiz React
        </h2>
        <span style={{
          backgroundColor: '#3498db',
          color: 'white',
          padding: '8px 15px',
          borderRadius: '20px',
          fontWeight: 'bold'
        }}>
          {questionActuelle + 1} / {questions.length}
        </span>
      </div>

      {/* Progression */}
      <div style={{
        height: '8px',
        backgroundColor: '#ecf0f1',
        borderRadius: '10px',
        marginBottom: '30px',
        overflow: 'hidden'
      }}>
        <div style={{
          height: '100%',
          width: `${((questionActuelle + 1) / questions.length) * 100}%`,
          backgroundColor: '#3498db',
          transition: 'width 0.3s'
        }} />
      </div>

      {/* Question */}
      <h3 style={{
```

```
        color: '#2c3e50',
        fontSize: '1.5rem',
        marginBottom: '30px',
        lineHeight: '1.6'
      }}>
        {question.question}
      </h3>

      {/* Options */}
      <div style={{ display: 'flex', flexDirection: 'column', gap: '15px' }}>
        {question.options.map((option, index) => (
          <button
            key={index}
            onClick={() => handleReponse(option)}
            style={{
              padding: '20px',
              backgroundColor: 'white',
              border: '2px solid #3498db',
              borderRadius: '10px',
              fontSize: '1.1rem',
              cursor: 'pointer',
              textAlign: 'left',
              transition: 'all 0.2s'
            }}
            onMouseEnter={(e) => {
              e.target.style.backgroundColor = '#3498db';
              e.target.style.color = 'white';
            }}
            onMouseLeave={(e) => {
              e.target.style.backgroundColor = 'white';
              e.target.style.color = '#2c3e50';
            }}
          >
            {String.fromCharCode(65 + index)}. {option}
          </button>
        ))}
      </div>
    </div>
  );
}

export default Quiz;
```

📖 Explications Détaillées

1. Gestion de l'index de question

```
const [questionActuelle, setQuestionActuelle] = useState(0);
```

On utilise un index pour savoir quelle question afficher. Cet index s'incrémente à chaque réponse.

2. Enregistrement des réponses

```
const nouvellesReponses = [...reponses, {
  question: questions[questionActuelle].question,
  reponseUtilisateur: option,
  reponseCorrecte: questions[questionActuelle].reponseCorrecte,
  correct: option === questions[questionActuelle].reponseCorrecte
}];
```

Chaque réponse est un objet contenant la question, la réponse de l'utilisateur, la bonne réponse, et un booléen **correct**.

3. Calcul du score

```
const calculerScore = () => {
  return reponses.filter(r => r.correct).length;
};
```

filter() garde seulement les réponses correctes, puis **.length** compte combien il y en a.

4. Affichage conditionnel

```
if (termine) {
  // Afficher l'écran de résultats
}
// Sinon afficher la question actuelle
```

On utilise le state **termine** pour basculer entre les deux vues : quiz en cours ou résultats.

> **✅ Résultat : Un quiz complet avec progression, score détaillé et récapitulatif des réponses !**

🎉 Félicitations !

Vous avez terminé les exercices 4-7 niveau intermédiaire ! Vous maîtrisez maintenant :

- Formulaires complexes avec validation
- Filtrage et recherche avancés
- Chronomètres avec useEffect
- Applications multi-étapes avec navigation

Fin de la Partie 2

Dans la Partie 3, nous passerons au niveau avancé avec APIs et projets complexes !