Guide du Formateur - Formation React

Vue d'ensemble

Ce guide vous accompagne dans l'animation de la formation React. Il contient :

- Les points bloquants fréquents et comment les résoudre
- Les erreurs courantes des débutants
- Les astuces pédagogiques
- Le déroulé recommandé

Déroulé recommandé

Jour 1 : Les bases (Module 1)

Durée	Contenu	Exercices
1h	Introduction JSX	ex01-carte-visite
1h	Composants et props	ex02, ex05
30min	Pause + Questions	
1h30	Listes et rendu conditionnel	ex03, ex07
1h	Introduction useState	ex04, ex08

Jour 2 : Module 1 suite + début Module 2

Durée	Contenu	Exercices
1h	Formulaires contrôlés	ex09
1h30	Mini-app de synthèse	ex10-todo
30min	Pause	
1h30	useEffect et cycle de vie	module-2/ex01, ex02
1h	Fetch et états loading/error	module-2/ex02

Jour 3 : Module 2 avancé + Projet

Durée	Contenu	Exercices
1h	useReducer	module-2/ex05
1h	Context API	module-2/ex09
30min	Pause	
2h+	Projet fil rouge TeamHub	Étapes 1-3

Jour 4 : Projet fil rouge

Durée	Contenu
4h+	TeamHub étapes 4-6

Durée	Contenu
	Révisions et questions

Points Bloquants Fréquents

1. JSX : Retourner plusieurs éléments

Erreur courante :

```
// ❌ ERREUR : Plusieurs éléments racine
function App() {
  return (
    <h1>Titre</h1>
    <p>Paragraphe</p>
  );
}
```

Explication pour l'apprenant :

> "En JSX, un composant doit retourner UN SEUL élément. Imaginez que vous devez mettre tout dans une seule boîte."

Solutions :

```
// Solution 1 : Envelopper dans une div
return (
  <div>
    <h1>Titre</h1>
    <p>Paragraphe</p>
  </div>
);

// Solution 2 : Fragment (pas d'élément HTML en plus)
return (
  <>
    <h1>Titre</h1>
    <p>Paragraphe</p>
  </>
);
```

---

2. La prop `key` dans les listes

Erreur courante :

```
Warning: Each child in a list should have a unique "key" prop.
```

Explication pour l'apprenant :

> "Quand vous utilisez `map()`, React a besoin d'un identifiant unique pour chaque élément. C'est comme donner un numéro à chaque élève : ça permet de savoir qui est qui même si quelqu'un change de place."

Règles :

1. Toujours utiliser `key` sur l'élément RACINE du map
2. Utiliser l'ID de l'objet si disponible
3. Ne PAS utiliser l'index si la liste peut changer

```
// ✅ CORRECT
{users.map(user => (
  <UserCard key={user.id} user={user} />
))}

// ⚠️ OK si liste statique
{items.map((item, index) => (
  <li key={index}>{item}</li>
))}

// ❌ MAUVAIS si la liste change
{users.map((user, index) => (
  <UserCard key={index} user={user} />  // Bug si suppression/ajout
))}
```

---

3. useState : Mise à jour asynchrone

Erreur courante :

```
const [count, setCount] = useState(0);

function handleClick() {
  setCount(count + 1);
  console.log(count);  // Affiche toujours l'ancienne valeur !
}
```

Explication pour l'apprenant :

> "setState ne change pas immédiatement la valeur. C'est comme envoyer un courrier : vous savez ce que vous avez écrit, mais le destinataire le recevra plus tard."

Solutions :

```
// Si besoin de l'ancienne valeur pour calculer la nouvelle :
setCount(prevCount => prevCount + 1);

// Pour voir le changement, utiliser useEffect :
useEffect(() => {
  console.log('count a changé :', count);
}, [count]);
```

---

4. useEffect : Boucle infinie

Erreur courante :

```
// ❌ BOUCLE INFINIE !
useEffect(() => {
  setData(fetchData());
});  // Pas de tableau de dépendances
```

Explication pour l'apprenant :

> "Sans tableau de dépendances, useEffect s'exécute à CHAQUE rendu. setData cause un re-rendu, qui déclenche useEffect, qui cause un re-rendu... à l'infini !"

Règles :

```
// S'exécute à CHAQUE rendu (rarement voulu)
useEffect(() => { ... });

// S'exécute UNE SEULE FOIS au montage
useEffect(() => { ... }, []);

// S'exécute quand `id` change
useEffect(() => { ... }, [id]);
```

---

5. useEffect : async dans useEffect

Erreur courante :

```
// ❌ ERREUR : useEffect ne peut pas être async
useEffect(async () => {
  const data = await fetch(...);
}, []);
```

Explication pour l'apprenant :

> "useEffect doit retourner soit rien, soit une fonction de cleanup. Une fonction async retourne une Promise, ce qui cause des problèmes."

Solution :

```
// ✅ CORRECT : Créer une fonction async à l'intérieur
useEffect(() => {
  async function fetchData() {
    const response = await fetch(...);
    const data = await response.json();
    setData(data);
  }

  fetchData();
}, []);
```

---

6. Formulaires : Input non contrôlé

Erreur courante :

```
Warning: A component is changing an uncontrolled input to be controlled.
```

Explication pour l'apprenant :

> "React veut savoir qui 'contrôle' l'input. Si vous passez de undefined à une valeur, React est confus."

Cause typique :

```
// ❌ Si user.name est undefined au départ
<input value={user.name} />
```

Solution :

```
// ✅ Toujours une valeur par défaut
<input value={user.name || ''} />

// Ou initialiser correctement le state
const [user, setUser] = useState({ name: '' });
```

---

7. Props : Oublier de passer une prop

Erreur courante :

```
// Dans le parent
<UserCard />  // Oups, pas de user !

// Dans UserCard
function UserCard({ user }) {
  return <p>{user.name}</p>;  // Error: Cannot read property 'name' of undefined
}
```

Solutions :

```
// Solution 1 : Valeur par défaut
function UserCard({ user = {} }) {
  return <p>{user.name || 'Inconnu'}</p>;
}

// Solution 2 : Vérification
function UserCard({ user }) {
  if (!user) return null;
  return <p>{user.name}</p>;
}

// Solution 3 : Optional chaining
function UserCard({ user }) {
  return <p>{user?.name || 'Inconnu'}</p>;
}
```

---

8. Context : Utilisé hors du Provider

Erreur courante :

```
Error: useAuth must be used within an AuthProvider
```

Explication pour l'apprenant :

> "Le contexte est comme un réseau WiFi : il faut être 'dans la zone' (à l'intérieur du Provider) pour y accéder."

Vérifier :

```
// main.jsx ou App.jsx
<AuthProvider>
  <App />  {/* Tout ce qui est ici peut utiliser useAuth */}
</AuthProvider>
```

Erreurs de Débutants par Concept

JSX

- Oublier d'importer React (versions < 17)
- Utiliser `class` au lieu de `className`
- Oublier de fermer les balises auto-fermantes (`<img>` → `<img />`)
- Utiliser `onclick` au lieu de `onClick` (camelCase)

Composants

- Nommer avec une minuscule (`userCard` → `UserCard`)
- Oublier l'export
- Confondre export default et export nommé

State

- Muter le state directement (`state.push(item)`)
- Ne pas utiliser le setter (`count = count + 1`)
- Oublier l'état initial de useState

Props

- Passer un objet en string (`user="Victor"` au lieu de `user={userObj}`)
- Oublier les accolades pour les valeurs JS

Listes

- Oublier la key
- Mettre la key sur le mauvais élément
- Utiliser l'index comme key sur une liste dynamique

Astuces Pédagogiques

1. Commencer par le résultat visuel

> "Voici ce qu'on veut obtenir... maintenant, décomposons."

Montrer d'abord l'interface finale, puis décomposer en composants.

2. Utiliser des analogies

Concept	Analogie
Components	Briques LEGO
Props	Paramètres d'une fonction
State	Mémoire du composant
useEffect	"Quand X change, fais Y"
Context	Variable globale accessible partout
key	Numéro de sécurité sociale

3. Live coding progressif

1. Écrire le squelette du composant
2. Ajouter le JSX statique
3. Remplacer les valeurs par des props/state
4. Ajouter les handlers
5. Refactoriser si nécessaire

4. Encourager les erreurs

> "Faisons exprès de casser le code pour voir l'erreur..."

Montrer les erreurs courantes aide à les reconnaître plus tard.

5. Utiliser React DevTools

Montrer comment inspecter :

- Les composants dans l'arbre
- Les props reçues
- L'état actuel
- Les re-renders (highlight updates)

Questions Fréquentes des Apprenants

"Pourquoi pas juste du JavaScript vanilla ?"

> "Vanilla JS fonctionne pour de petits projets. Mais quand l'interface devient complexe, synchroniser le DOM avec les données devient un cauchemar. React s'en occupe pour vous."

"C'est quoi le Virtual DOM ?"

> "Imaginez une maquette de votre maison. Au lieu de casser et reconstruire chaque pièce, vous modifiez la maquette, puis appliquez uniquement les changements nécessaires à la vraie maison."

"Quand utiliser useState vs useReducer ?"

> "useState pour les cas simples (un booléen, une string). useReducer quand vous avez plusieurs valeurs liées ou des logiques de mise à jour complexes."

"C'est quoi la différence entre props et state ?"

> "Props = ce qu'on vous donne (immutable). State = ce que vous possédez (modifiable). Les props viennent du parent, le state est interne au composant."

Checklist de Fin de Formation

L'apprenant sait :

- ☐ Créer un composant fonctionnel
- ☐ Utiliser JSX correctement
- ☐ Passer et utiliser des props
- ☐ Gérer l'état avec useState
- ☐ Itérer avec map() et utiliser key
- ☐ Faire du rendu conditionnel
- ☐ Créer des formulaires contrôlés
- ☐ Utiliser useEffect pour les effets de bord
- ☐ Charger des données avec fetch
- ☐ Gérer loading/error states
- ☐ Utiliser useReducer pour les états complexes
- ☐ Créer et utiliser un Context
- ☐ Créer un hook personnalisé

Bonus :

- ☐ Comprendre le cycle de vie
- ☐ Utiliser les DevTools
- ☐ Optimiser avec useMemo/useCallback
- ☐ Structurer un projet React

Ressources Complémentaires

Documentation officielle

- React.dev - Nouvelle documentation officielle
- React.dev/learn - Tutoriel interactif

Outils

- React DevTools (extension navigateur)
- VS Code + extensions ESLint, Prettier

Pour aller plus loin

- React Router pour le routing
- Redux / Zustand pour le state management
- React Query / SWR pour le data fetching
- Next.js / Remix pour le SSR