Patterns avancés

HOC, render props et custom hooks

- Custom Hooks
- HOC et Render Props

Custom Hooks

Custom Hooks

Un custom hook est une fonction commençant par `use` qui encapsule de la logique réutilisable avec des hooks React.

useLocalStorage

```
function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    try {
      const stored = localStorage.getItem(key);
      return stored ? JSON.parse(stored) : initialValue;
    } catch { return initialValue; }
  });

  const setStoredValue = (newValue) => {
    setValue(newValue);
    localStorage.setItem(key, JSON.stringify(newValue));
  };

  return [value, setStoredValue];
}

// Usage
const [theme, setTheme] = useLocalStorage('theme', 'light');
```

useFetch

```
function useFetch(url) {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
```

```
  useEffect(() => {
    const controller = new AbortController();
    setLoading(true);

    fetch(url, { signal: controller.signal })
      .then(res => { if (!res.ok) throw new Error(res.statusText); return res.json(); })
      .then(data => { setData(data); setLoading(false); })
      .catch(err => { if (err.name !== 'AbortError') { setError(err); setLoading(false); } });

    return () => controller.abort();
  }, [url]);

  return { data, loading, error };
}

// Usage
const { data: users, loading } = useFetch('/api/users');
```

useDebounce

```
function useDebounce(value, delay = 500) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const timer = setTimeout(() => setDebouncedValue(value), delay);
    return () => clearTimeout(timer);
  }, [value, delay]);

  return debouncedValue;
}

// Usage : recherche sans spammer l'API
function Recherche() {
  const [query, setQuery] = useState('');
  const debouncedQuery = useDebounce(query, 300);

  const { data } = useFetch(`/api/search?q=${debouncedQuery}`);
}
```

HOC et Render Props

Higher-Order Components (HOC)

Un HOC est une fonction qui prend un composant et retourne un nouveau composant enrichi. Pattern issu des classes, encore utilisé pour des wrappers génériques.

```
// withAuth — HOC de protection de route
function withAuth(WrappedComponent) {
  return function AuthenticatedComponent(props) {
    const { user } = useAuth();

    if (!user) return <Navigate to="/login" />;
    return <WrappedComponent {...props} user={user} />;
  };
}

// Usage
const ProtectedDashboard = withAuth(Dashboard);

// withLoading — HOC d'affichage de chargement
function withLoading(WrappedComponent) {
  return function ({ isLoading, ...props }) {
    if (isLoading) return <Spinner />;
    return <WrappedComponent {...props} />;
  };
}
```

Render Props

Pattern où un composant reçoit une fonction en prop et l'appelle pour déléguer le rendu. Remplacé en grande partie par les custom hooks.

```
// DataFetcher avec render prop
function DataFetcher({ url, render }) {
```

```
  const { data, loading, error } = useFetch(url);
  return render({ data, loading, error });
}

// Usage
<DataFetcher
  url="/api/users"
  render={({ data, loading }) => (
    loading ? <Spinner /> : <UserList users={data} />
  )}
/>

// Version children-as-function
<DataFetcher url="/api/users">
  {({ data, loading }) => loading ? <Spinner /> : <UserList users={data} />}
</DataFetcher>
```

Recommandation actuelle

Dans React moderne, les **custom hooks** couvrent la majorité des cas d'usage des HOC et render props avec moins de complexité et de nesting. Utiliser HOC/render props principalement pour l'intégration avec des bibliothèques tierces ou la compatibilité legacy.