Ministry of SMEs and Startups

KOSME Korea SMEs and Startups Agency

LIKELION

JOBKOREA

LESSON 08
Context API & Global State Management

WEEK 02

Introduction to Context API

❖ What is Context API?

- ➢ A React feature for sharing data globally without prop drilling.

❖ Why use it?

- ➢ Simplifies state management for deeply nested components.

❖ Official Reference

- ➢ React Context API

Prop Drilling Problem

❖ What is Prop Drilling?

- ➢ Passing props through multiple component layers to reach a nested component.

❖ Issues with Prop Drilling

- ➢ Makes code harder to maintain and read.

```tsx
export default function PropDrilling() {
  const [theme, setTheme] = useState('light');
  return (
    <div>
      <Middle theme={theme} />

      <button
        onClick={() => {
          setTheme(theme === 'light' ? 'dark' : 'light');
        }}
      >
        Toggle Theme
      </button>
    </div>
  );
}

type MiddleProps = { theme: string };
function Middle({ theme }: MiddleProps) {
  return <Child theme={theme} />;
}

function Child({ theme }: MiddleProps) {
  return <div>{theme}</div>;
}
```

How Context API Solves Prop Drilling

- **Direct Data Access**
  - Context allows components to access data without intermediate props.
- **Key Benefit**
  - Cleaner, more scalable code structure.
- **Basic Flow**
  - Create context → Provide data → Consume data.

Creating a Context

- **Using createContext**
  - const MyContext = React.createContext(defaultValue);
- **Default Value**
  - Used when no Provider is found in the tree.

```
1  const ThemeContext = createContext<string>('light');
```

Providing Context

❖ **Context.Provider**

Wraps components to provide context data.

❖ **Syntax**

<MyContext.Provider value={value}>

```
export default function ContextAPI() {
  const [theme, setTheme] = useState('light');
  return (
    <div>
      <h5>Context API Example</h5>
      <ThemeContext.Provider value={theme}>
        <button
          onClick={() => {
            setTheme(theme === 'light' ? 'dark' : 'light');
          }}
        >
          Toggle Theme
        </button>
        <Child />
        <p>Current Theme: {theme}</p>
      </ThemeContext.Provider>
    </div>
  );
}
```

Consuming Context

❖ Using useContext Hook

- ➢ Simplest way to access context in functional components.

❖ Syntax

- ➢ const value = useContext(MyContext);

```
1  function Child() {
2    const theme = useContext(ThemeContext);
3    return <div>{theme}</div>;
4  }
```

When to Use Context API

- **Ideal Use Cases**
  - Theme switching, user authentication, localization.
- **Not for Everything**
  - Avoid for frequent updates or complex state logic.
- **Guideline**
  - Use for stable, global data.

Context API Example: Theme Toggle

- **Scenario**
- Toggle between light and dark themes.

```jsx
import { createContext, useContext, useState } from 'react';

const ThemeContext = createContext<string>('light');

export default function ContextAPI() {
  const [theme, setTheme] = useState('light');
  return (
    <div>
      <h5>Context API Example</h5>
      <ThemeContext.Provider value={theme}>
        <button
          onClick={() => {
            setTheme(theme === 'light' ? 'dark' : 'light');
          }}
        >
          Toggle Theme
        </button>
        <Child />
        <p>Current Theme: {theme}</p>
      </ThemeContext.Provider>
    </div>
  );
}

function Child() {
  const theme = useContext(ThemeContext);
  return <div>{theme}</div>;
}
```

Splitting Contexts

❖ Why Split?

Avoids unnecessary re-renders when unrelated data changes.

```jsx
import { useState, createContext, useContext } from 'react';

const ThemeContext = createContext('light');
const UserContext = createContext('Guest');

export default function SplittingContexts() {
  const [theme, setTheme] = useState('light');
  const [user, setUser] = useState('Guest');
  return (
    <ThemeContext.Provider value={theme}>
      <UserContext.Provider value={user}>
        <Component />
      </UserContext.Provider>
    </ThemeContext.Provider>
  );
}

function Component() {
  const theme = useContext(ThemeContext);
  const user = useContext(UserContext);

  return (
    <div>
      <p>Current Theme: {theme}</p>
      <p>Current User: {user}</p>
    </div>
  );
}
```

Context with TypeScript

❖ **Typed Context**

Ensures type safety for context values.

```tsx
import { createContext } from 'react';

interface ThemeContextType {
  theme: string;
  setTheme: (theme: string) => void;
}

const ThemeContext = createContext<ThemeContextType | undefined>(undefined);

export default function ContextWithTypeScript() {
  return (
    <ThemeContext.Provider value={{ theme: 'light', setTheme: () => {} }}>
      <div>ContextWithTypeScript</div>
    </ThemeContext.Provider>
  );
}
```

Custom Hooks with Context

❖ Why Custom Hooks?

Encapsulates context logic for reusability.

```
function Component() {
  const { theme, setTheme } = useTheme();
  return <p>Theme: {theme}</p>;
}

function useTheme() {
  const context = useContext(ThemeContext);
  if (!context) throw new Error('useTheme must be used within ThemeProvider');
  return context;
}
```

Context API Limitations

- **Not for Complex State**
  - Lacks built-in middleware, dev tools like Redux.
- **Performance**
  - Can cause re-renders without optimization.
- **Solution**
  - Use Redux/Zustand for large apps.

Global State Management Overview

❖ What is Global State?

- ➢ State shared across multiple components.

❖ Tools

- ➢ Context API, Redux, Zustand, Recoil.

❖ Context API’s Role

- ➢ Lightweight solution for simple global state.

Zustand as an Alternative

❖ What is Zustand?

- ➢ Lightweight state management library.

❖ Benefits

- ➢ Simpler than Redux, no boilerplate.

```
1
2  type BearState = {
3    bears: number;
4    increasePopulation: () => void;
5    removeAllBears: () => void;
6  };
7
8  const useBearStore = create<BearState>((set) => ({
9    bears: 0,
10   increasePopulation: () =>
11     set((state) => {
12       return { bears: state.bears + 1 };
13     }),
14   removeAllBears: () => set({ bears: 0 }),
15 }));
16
17 function BearCounter() {
18   const bears = useBearStore((state) => state.bears);
19   return <h1>{bears} around here ...</h1>;
20 }
21
22 function Controls() {
23   const increasePopulation = useBearStore((state) => state.increasePopulation);
24   return <button onClick={increasePopulation}>one up</button>;
25 }
26
```