Using Context and Custom React Hooks for a Redux Like Experience

Peter Kellner

Developer, Consultant and Author

ReactAtScale.com @pkellner linkedin.com/in/peterkellner99

Theme Context Built in the Previous Module

App.js

```
import { createContext, useState } from "react";
import Layout from "./components/layout/Layout";

export const ThemeContext = createContext({});

const App = ({ url }) => {
  const [darkTheme, setDarkTheme] = useState(false);
  const toggleTheme = () => setDarkTheme(!darkTheme);

  const value = {
    darkTheme,
    toggleTheme,
  };

  return (
    <ThemeContext.Provider value={value}>
      <Layout url={url} />
    </ThemeContext.Provider>
  );
};

export default App;
```

Separating UI From Theme Management

```
export const ThemeContext = createContext({});

function App() {
  const [darkTheme, setDarkTheme] = useState(false);
  const toggleTheme = () =>
    setDarkTheme(!darkTheme);

  const value = {
    darkTheme,
    toggleTheme,
  };

  return (
    <ThemeContext.Provider value={value}>
      <UI />
    </ThemeContext.Provider>
  );
};
```

Separating UI From Theme Management

```
export const ThemeContext =
  createContext({});

function App() {
  const [darkTheme, setDarkTheme] =
  useState(false);
  const toggleTheme = () =>
    setDarkTheme(!darkTheme);
  const value = {
    darkTheme,
    toggleTheme,
  };
  return (
    <ThemeContext.Provider value={value}>
      <UI />
    </ThemeContext.Provider>
  );
};
```

Separating UI From Theme Management

```
export const ThemeContext =
  createContext({});

function App() {
  const [darkTheme, setDarkTheme] =
  useState(false);
  const toggleTheme = () =>
    setDarkTheme(!darkTheme);
  const value = {
    darkTheme,
    toggleTheme,
  };
  return (
    <ThemeContext.Provider value={value}>
      <UI />
    </ThemeContext.Provider>
  );
};
```

```
function App() {
  return (
    <ContextAndStateManager>
      <UI />
    </ContextAndStateManager>
  );
};
```

Separating UI From Theme Management

```
export const ThemeContext =
  createContext({});

function App() {
  const [darkTheme, setDarkTheme] =
  useState(false);
  const toggleTheme = () =>
    setDarkTheme(!darkTheme);
  const value = {
    darkTheme,
    toggleTheme,
  };
  return (
    <ThemeContext.Provider value={value}>
      <UI />
    </ThemeContext.Provider>
  );
};
```

```
function App() {
  return (
    <ContextAndStateManager>
      <UI />
    </ContextAndStateManager>
  );
};
```

```
export const ThemeContext = createContext({});
function ContextAndStateManager({children}) {
  const [darkTheme, setDarkTheme] = "..."
  const toggleTheme = () => { return "..."}
  return (
    <ThemeContext.Provider
        value={{darkTheme, toggleTheme}}>
      {children}
    </ThemeContext.Provider>
  );
}
```

Problems with Wrapping App Component with a Context Provider

Code base confusing with global variables everywhere

Any context reference in a component forces the component to re-render on any change

Was Refactoring Out a Theme Component a Good Idea?

Likely no	VS	Maybe yes but ...
Not reusing useTheme		Separated concerns
Logic is very simple		Kept functionality in logical components

Demo

More examples of shared context

- Make speaker delete button work
- Make speaker add button work

Demo

Learn about custom React hooks

- Add a general-purpose custom React Hook that manages calls to a REST server
- Add a more specialized custom React hooks for speaker's data that calls the general-purpose custom hook.

Demo

Learn more about sharing React contexts

- Leverage useContext for speaker landing pages like /speaker/1124
- Handle case of having Context defined in same JavaScript file as useContext being referenced

```
import { SpeakersDataProvider } from "../contexts/SpeakersDataContext";
import { SpeakerMenuProvider } from "../contexts/SpeakerMenuContext";

function Speakers() {
  const { darkTheme } = useContext(ThemeContext);

  return (
    <div className={darkTheme ? "theme-dark" : "theme-light"}>
      <SpeakerMenuProvider>
        <SpeakerMenu />
        <div className="container">
          <div className="row g-4">
            <SpeakersDataProvider>
              <SpeakersList />
            </SpeakersDataProvider>
          </div>
        </div>
      </SpeakerMenuProvider>
    </div>
  );
}
```

```
import { SpeakersDataProvider } from "../contexts/SpeakersDataContext";
import { SpeakerMenuProvider } from "../contexts/SpeakerMenuContext";

function Speakers() {
  const { darkTheme } = useContext(ThemeContext);

  return (
    <div className={darkTheme ? "theme-dark" : "theme-light"}>
      <SpeakerMenuProvider>
        <SpeakerMenu />
        <div className="container">
          <div className="row g-4">
            <SpeakersDataProvider>
              <SpeakersList />
            </SpeakersDataProvider>
          </div>
        </div>
      </SpeakerMenuProvider>
    </div>
  );
}
```

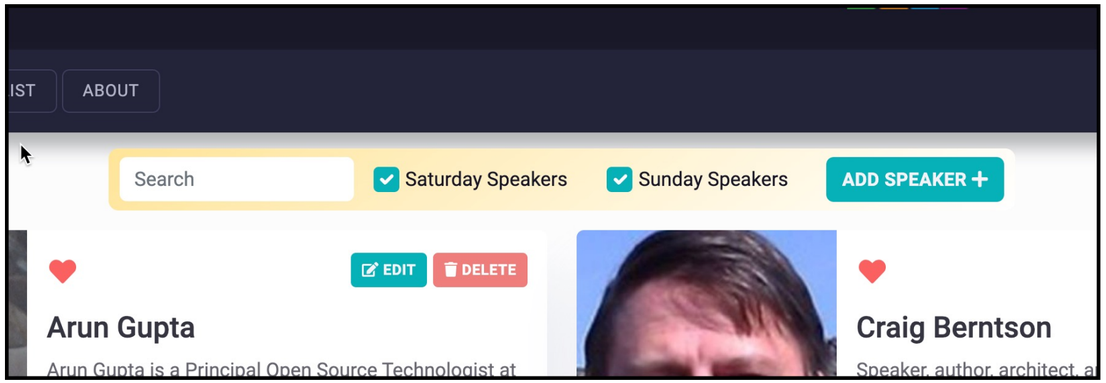
```
import { SpeakersDataContext } from "../contexts/SpeakersDataContext";
import useSpeakerSortAndFilter from "../hooks/useSpeakerSortAndFilter";

export default function SpeakersList() {
  const { speakerList, loadingStatus } = useContext(SpeakersDataContext);
  const speakerListFiltered = useSpeakerSortAndFilter(speakerList);
  if (loadingStatus === "loading") {
    return <div className="card">Loading...</div>;
  }
  return (
    <>
      {speakerListFiltered.map(function (speakerRec) {
        return (
          <SpeakerDetail
            key={speakerRec.id}
            speakerRec={speakerRec}
            showDetails={false}
          />
        );
      })}
    </>
  );
}
```

Demo

Implement a speaker menu with shared context

- Handle multiple criteria
- Learn about UI pivot menus



Takeaways

Learned how combining React hooks together can significantly enhance your apps

Specifically, combining React's useContext with other hooks (useState) is very powerful

- Apps are easy to breakdown into building blocks
- Building blocks easy to integrate and reuse
- Improved React app maintenance and updates