React Context

Prior Knowledge

- Be able to manage state in React.
- Be able to pass props in React.

Learning Objectives

- Understand how to create a React context.
- Understand how to provide values to a context.
- Understand how to consume value from a context using `useContext`.
- Understand when it is appropriate to introduce context.

Context API

When passing information between components in React, we hold our values in `state` and pass them between components using `props`. In complex applications we can have state that needs to be used by several components at various levels of nesting in our component trees. This invariably leads to state being held very high (often in `App`) and passed through a long chain of props. React offers a solution to this called the Context API

The Context API is designed for state that needs to be shared by multiple components, or for state that can be considered `global`. For this reason it is **not** a wholesale replacement for props and you should be quite selective when deciding what contexts to use.

Common candidates for contexts are logged in users, themes or language settings. An example of a theme to switch between light and dark mode could be implemented as below.

```
// src/contexts/Theme.jsx
import { createContext } from 'react';

export const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

// src/App.jsx
const App = () => {
  return <div>{/* Rest of the app including <ToggleTheme /> */}</div>;
};

// src/main.jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ThemeProvider } from './contexts/Theme';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <ThemeProvider>
    <App />
  </ThemeProvider>
);

// src/components/ToggleTheme.jsx
import { useContext } from 'react';
import { ThemeContext } from '../contexts/Theme';

const ToggleTheme = () => {
  const { theme, setTheme } = useContext(ThemeContext);

  const toggleTheme = () => {
    setTheme((currTheme) => {
      return currTheme === 'light' ? 'dark' : 'light';
    });
  };

  return (
    <button onClick={toggleTheme} className={`button__${theme}`}>
      Change theme
    </button>
  );
};

export default ToggleTheme;
```

Here we can share the state of theme from App using a context provider and access the value using `useContext`. We would then have different CSS classNames for `App__light`, `App__dark` etc to change the look of our site.

Creating Contexts

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

export const ThemeContext = createContext();
```

Contexts can be created using `React.createContext()`. Create context can be passed an initial value to use when there is no Provider. This can be useful when writing front end tests for our components. As this is a topic for another day we can leave ours blank as we will always have a Provider.

Context.Provider

Once we have a context we can provide a value to it from one of our components. We pass a value prop to the `Context.Provider` and this value will be available to all the children of that provider. For this reason we should wrap our application in the Providers at a high level to ensure the values are available everywhere.

```
<ThemeContext.Provider value={{ theme, setTheme }}>
  {/* Rest of the app */}
</ThemeContext.Provider>
```

This can be done anywhere in the app but a typical good pattern is to create a dedicated `Provider` component that will hold any necessary state and wrap the whole App.

```
// src/contexts/Theme.jsx
import { createContext } from 'react';

export const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

// src/App.jsx
const App = () => {
  return <div>{/* Rest of the app */}</div>;
};

// src/main.jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ThemeProvider } from './contexts/Theme';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <ThemeProvider>
    <App />
  </ThemeProvider>
);
```

nb The `ThemeProvider` component is rendering it's `children`, which in this case is `<App />`. For more information on this see the `composition in react` notes.

useContext

To access the value of the provider we have the `useContext` hook. This hook will return the value from the first parent Provider. (Typically there will only be one parent provider but it is possible to have multiple). We can then use those values just as if they had been passed on props and our component will re-render when those values are updated.

For example, if we wanted to toggle the theme between light and dark we would set the theme state just as we would set a local state. An implementation could look something like below:

```
// src/Components/ToggleTheme.jsx
import { useContext } from 'react';
import { ThemeContext } from '../contexts/Theme';

const ToggleTheme = () => {
  const { theme, setTheme } = useContext(ThemeContext);

  const toggleTheme = () => {
    setTheme((currTheme) => {
      return currTheme === 'light' ? 'dark' : 'light';
    });
  };

  return (
    <button onClick={toggleTheme} className={`button__${theme}`}>
      Change theme
    </button>
  );
};

export default ToggleTheme;
```