React Styled Components

Lernziele

- [ ] Verstehen, was eine CSS-in-JS-Bibliothek ist und warum wir sie normalen CSS vorziehen
- [ ] Wissen, wie man Styled Components verwendet
    - [ ] einfache Styled Components erstellen
    - [ ] benutzerdefinierte Komponenten stylen
    - [ ] Styling basierend auf Props anpassen
    - [ ] verschachtelte Styles mit Pseudo-Elementen und Pseudo-Klassen verwenden
    - [ ] globale Styles schreiben
- [ ] Wissen, wie man Schriften von Google mit Next.js verwendet

---

Was ist CSS-in-JS und warum verwenden wir es?

CSS-in-JS bezieht sich auf eine Sammlung von Ideen, um komplexe Probleme in CSS zu lösen. Es gibt mehrere Bibliotheken, die diesen Ansatz verwenden, eine davon sind **Styled Components**. Alle Implementierungen haben gemeinsam, dass sie JavaScript als Sprache zur Erstellung von Styles verwenden.

Hier ist eine Liste von Vorteilen einer CSS-in-JS-Bibliothek wie **Styled Components**:

- Automatisch nur das kritische CSS wird injiziert (und nichts weiter)
- Keine Klassennamenfehler
- Einfacheres Löschen von CSS
- Einfaches dynamisches Styling
- Wartung ohne Kopfschmerzen
- Automatisches Hinzufügen von Vendor-Prefixes

> 📙 Lies mehr über die Motivation zur Verwendung von Styled Components.

Styling mit Styled Components

Basis Styling

Um eine Styled Component zu erstellen,

- importiere `styled`
- verwende es, um eine Styled Component wie `ListItem` zu erstellen, und
- implementiere die Styled Component im Return-Statement deiner Komponente.

```
//components/List.js
import styled from "styled-components";
```

```
export default function List() {
  return (
    <StyledList>
      <ListItem>Item 1</ListItem>
      <ListItem>Item 2</ListItem>
      <ListItem>Item 3</ListItem>
    </StyledList>
  );
}

const ListItem = styled.li`
  background-color: crimson;
`;

const StyledList = styled.ul`
  list-style-type: none;
`;
```

> 💡 Beachte, dass der Name einer Styled Component groß geschrieben ist (weil es eine Komponente ist), aber nicht mit dem Funktionsnamen übereinstimmen muss; ein üblicher Namensstil ist das Hinzufügen des Wortes `Styled`.

> 📙 Lies mehr über Basic Styling mit Styled Components.

Styling einer benutzerdefinierten Komponente

Manchmal gibt es bereits eine Komponente mit vordefinierten Styles, die du aber erweitern möchtest:

- Du hast vielleicht eine `Button`-Komponente mit grundlegenden Styles selbst definiert siehe ein gutes Beispiel in den Docs, oder
- Ein Framework bietet eine Komponente, die du verwenden möchtest, wie die Link-Komponente von next/link:

```
import styled from "styled-components";
import Link from "next/link";

export default function List() {
  return (
    <ul>
      <li>
        <StyledLink>Let's go somewhere!</StyledLink>
      </li>
    </ul>
  );
}

const StyledLink = styled(Link)`
  text-decoration: none;
`;
```

> 📙 Lies mehr über das Styling jeder Komponente.

Anpassung basierend auf Props

Du kannst das Styling basierend auf Props anpassen. Dazu musst du die Props an die Styled Component übergeben. Meistens möchtest du das Prop mit einem `$`-Prefix versehen. Dies zeigt Styled Components an, dass das Prop nicht an das zugrunde liegende DOM-Element oder die Komponente übergeben, sondern nur für das Styling verwendet werden soll.

```
import styled from "styled-components";

export default function List() {
  return (
    <StyledList $isOnFire>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </StyledList>
  );
}
```

Um die Props zu verwenden, um die Styles zu ändern, interpolierst du eine Funktion in das Styling-Template-String. Die Funktion erhält die Props als Argument.

Zum Beispiel kannst du den ternären Operator verwenden, um zu prüfen, ob eine Eigenschaft wahr oder falsch ist:

```
const StyledList = styled.ul`
  list-style-type: ${(props) => (props.$isOnFire ? "🔥" : "❄")};
  /* oder mit Destructuring: */
  list-style-type: ${({ $isOnFire }) => ($isOnFire ? "🔥" : "❄")};
`;
```

Wenn du mehrere CSS-Eigenschaften basierend auf demselben Prop setzen möchtest, kannst du den `css`-Helper verwenden:

```
import styled, { css } from "styled-components";

const StyledList = styled.ul`
  ${({ $isOnFire }) =>
    $isOnFire &&
    css`
      list-style-type: "🔥";
      background-color: red;
      color: white;
    `}
`;
```

> 💡 Neben anderen Vorteilen bietet der `css`-Helper Syntax-Highlighting und Performance-Optimierung.

> 📙 Lies mehr über das Styling basierend auf Props.

Pseudoelemente und Pseudoselektoren

Um Pseudoelemente, Pseudoselektoren oder verschachtelte Styles anzuwenden, kannst du ein einzelnes Kaufmannsund `&` verwenden, das sich auf die Komponente selbst bezieht:

```
const StyledLink = styled(Link)`
  text-decoration: none;
  &:hover {
    color: red;
  }
`;
```

> 📙 Lies mehr über Pseudoelemente und Pseudoselektoren.

Globales Styling

Um globales Styling zu implementieren, musst du eine globale Styled Component erstellen. Um die Struktur des Projekts übersichtlich zu halten, erstelle eine `styles.js`-Datei im Stammverzeichnis des Projekts:

```
// styles.js
import { createGlobalStyle } from "styled-components";

export default createGlobalStyle`
  *,
  *::before,
  *::after {
    box-sizing: border-box;
  }
  body {
    margin: 0;
    font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
Helvetica,
      Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe UI
Symbol";
  }
  // Weitere globale Styles hier...
`;
```

Importiere die `GlobalStyle`-Komponente in die `pages/_app.js`-Datei und rendere sie über dem `<Component />`:

```
// pages/_app.js
import GlobalStyle from "../styles";

export default function App({ Component, pageProps }) {
  return (
    <>
      <GlobalStyle />
      <Component {...pageProps} />
    </>
  );
}
```

> 💡 Es gibt keinen Konsens darüber, wo die `GlobalStyle`-Komponente platziert werden sollte. Die Entscheidung für eine `styles.js`-Datei spiegelt wider, dass bis jetzt globale Styles in einer `styles.css`-Datei geschrieben wurden.

> 📙 Lies mehr über createGlobalStyle.

DSGVO-konforme Integration von Google Fonts

Next.js bietet das `@next/font`-npm-Paket. Es optimiert deine Schriften automatisch (einschließlich benutzerdefinierter Schriften) und entfernt externe Netzwerkaufrufe für verbesserte Privatsphäre und Leistung, indem Google-Schriften selbst gehostet werden.

Du musst zuerst `@next/font` in deinem Projekt installieren. Um eine Schriftfamilie zu implementieren, importiere sie, wo nötig, und verwende sie innerhalb der Styled Component.

Das folgende Beispiel setzt die Schriftfamilie in der `GlobalStyle`-Komponente für das HTML-`body`-Element:

```
import { createGlobalStyle } from "styled-components";
import { Open_Sans } from "@next/font/google";

const openSans = Open_Sans({ subsets: ["latin"] });

export default createGlobalStyle`
  // ... einige globale Styles hier...
  }
  body {
    margin: 0;
    font-family: ${openSans.style.fontFamily};
    padding: 2rem;
  }
    // ... einige weitere globale Styles hier ...
`;
```

> 📙 Lies mehr über Google Fonts in Next.js und schaue dir die API-Referenz für `@next/font` an.

Resources

- What actually is CSS-in-JS?
- styled components
- Next.js: Font Optimization
- Google Fonts