




# 4

## DS + styled-components with TS

 강의자	 조민지
 일자	@November 18, 2023

 DS

DS란?

DS는 왜 필요한가요?

 styled-components로 theme 만들기 with TS

[theme.ts](#)

[ThemeProvider](#)

[styled.d.ts](#)

같이 해볼까요?

[Code](#)

 styled-components 가변 스타일링

[Example in JS](#)

[TypeScript에선?](#)

 피그마보고 코드 설계하기

같이 해봐요 🙌

[Code](#)

 과제 - 오늘 실습에서 끝내지 못한 디자인 적용

모두 고생하셨습니다 🥰



### DS

#### ▼ DS란?

Design System

‘서비스의 목적에 맞도록 **일관되게** 구성된 일련의 패턴과 공유된 규칙 언어’,  
‘웹/모바일 인터페이스 디자인 시 **반복되어 활용**되는 컴포넌트나 리소스 등을 모아 만든 일종의 가이드라인’,  
‘**디자인 원칙**과 규격, **재사용**할 수 있는 UI 패턴과 컴포넌트 코드를 포괄하는 시스템’

§...? 이런 문장들 말고 직접 익혀봐요!

**bold체** 키워드만 일단 살펴보고 여기로 같이 가봐요 🗺️

[참고 레퍼런스](#)

#### ▼ DS는 왜 필요한가요?

조금은 감이 오셨나요?

대표적인 DS의 종류는 이렇게 됩니다.

- color
- typography
- component
- etc.

여기 잘 정리된 DS의 예를 보여줄게요!

Gmarket Design System

Line Design System

함께 디자인하는 디자이너간,  
함께 개발하는 개발자간,  
그리고 디자이너와 개발자 사이에서 일관된 규칙으로 소통할 수 있고, 불필요한 반복을 줄일 수 있게 됩니다. 변경사항이 있을 경우 수정에도 용이하며 개발자 입장에서 코드의 중복을 줄일 수 있죠.

## styled-components로 theme 만들기 with TS

### ▼ theme.ts

프로젝트에서 사용할 공통적인 스타일 속성을 해당 파일에 정의합니다.

대표적으로 DS에서의 color, typography가 `theme`에 속하게 됩니다.

```
// theme.ts
import { css } from 'styled-components';

const color = {
  white: 'ffffff';
  ...
}

const typography = {
  body1: () => css`
    font-size: 13px;
    font-weight: 500;
    line-height: 20px;
  `,
  title2: ({ fontWeight }: { fontWeight?: 'semiBold' | 'bold' }) => css`
    font-size: 19px;
    font-weight: fontWeight === 'bold' ? 700 : 600;
  `,
  ...
}

export const theme = {
  color,
  typography,
};
```

이 theme에 있는 color, typography의 type 추론은 어떻게 할 수 있을까요?

```
interface Color {
  white: string;
  red: string;
  ...
}
```

이거면 타입 추론도 가능하겠네요!

근데 이렇게 된다면 color가 추가될 때마다 수기로 type도 하나씩 추가해줘야해요.

조금 더 효율적인 방식은 없을까요? (답은 조금 이따 실습 때 같이 찾아 보아요 ☺)

### ▼ ThemeProvider

```
// App.tsx

import { ThemeProvider } from 'styled-components';
import { theme } from './styles/theme

const App = () => {
  return (
    <ThemeProvider theme={theme}>
      ...
    </ThemeProvider>
  );
}
```

이렇게 루트에 `ThemeProvider`를 감싸준다면 `ThemeProvider`는 `context`를 통해 하위의 모든 리액트 컴포넌트에게 **`theme`** 속성을 전달해줍니다.

## ▼ styled.d.ts

d.ts 파일은 **TS코드의 타입 추론**을 돕는 **타입 선언 파일**입니다.

```
// styled.d.ts

import 'styled-components';

declare module 'styled-components' {
  export interface DefaultTheme {
    Color;
    Typography;
  }
}
```

styled-components의 `DefaultTheme`에 인터페이스를 지정해줘야하므로 `interface`의 이름은 `DefaultTheme`로 해줍니다.

```
// theme.ts

import { DefaultTheme } from 'styled-components';

export const Theme : DefaultTheme {
  ...
}
```

이렇게 적용시켜주면 됩니다. 꼭 제대로 type 추론이 되는지 test 해보세요!

## ▼ 같이 해볼까요?

잠깐! `cra`로 만든 프로젝트에서 **불필요한 파일들**을 정리해주세요.

`index.css`, `App.css`와 같은 파일에 정의된 스타일이 디자인적으로 의도한 스타일이 아니라면 과감히 삭제해주세요 :)

`yarn add styled-components` 부터 먼저 해주세요!

이번 세션을 위해 **퀵해빈**님께서 피그마 디자인 작업을 맡아주셨습니다 🙏  
끝까지 열심히 해주실 거죠?

[https://www.figma.com/file/kPZaJS1lthdpYniZl0kZcp/%EB%A6%AC%EC%95%A1%ED%8A%B8-%EC%84%B8%EC%85%98\\_%EB%94%94%EC%9E%90%EC%9D%B8?type=design&node-id=0%3A1&mode=dev](https://www.figma.com/file/kPZaJS1lthdpYniZl0kZcp/%EB%A6%AC%EC%95%A1%ED%8A%B8-%EC%84%B8%EC%85%98_%EB%94%94%EC%9E%90%EC%9D%B8?type=design&node-id=0%3A1&mode=dev)

## ▼ Code

```
// theme.ts

import { css, DefaultTheme } from 'styled-components';

const color = {
  green: '#31F396',
  red: '#FF6868',
  gray1: '#51515C',
  gray2: '#AFAFBC',
  gray3: '#DEDEE8',
};

export type Color = typeof color;

const typography = {
  title1: css`
    font-size: 20px;
    font-weight: 700;
    line-height: 26px;
  `,
  body1: css`
    font-size: 16px;
    font-weight: 400;
    line-height: 20px;
  `,
};

export type Typography = typeof typography;

export const theme: DefaultTheme = {
  color,
  typography,
};
```

```
// App.tsx

import { ThemeProvider } from 'styled-components';
import { theme } from './ds/theme';

function App() {

  return (
    <ThemeProvider theme={theme}>
      ...
    </ThemeProvider>
  );
}

export default App;
```

```
// styled.d.ts

import 'styled-components';
import { Color, Typography } from './theme';

declare module 'styled-components' {
  export interface DefaultTheme {
    color: Color;
    typography: Typography;
  }
}
```

```
}  
}
```

## styled-components 가변 스타일링

### ▼ Example in JS

가변 스타일링을 하고 싶을 땐 **props**를 사용하는 건 기억하시죠~?

```
// in JavaScript  
  
import styled from 'styled-components';  
  
export const MenuBar = ( color, isGap ) => {  
  <Button $color={color} $isGap={isGap}>버튼</Button>  
}  
  
const Button = styled.button`  
  ...  
  color: ${({ $color }) => $color || '#000000' };  
  gap: ${({ $isGap }) => $isGap ? '10px' : 0 };  
  ...  
`;  
;
```

왜 props명 앞에 \$를 붙이나요?

하지만 \$ prefix를 붙이는 것에 있어서도 호의적인 입장과 그렇지 않은 입장이 함께 있다고 합니다 :)

### ▼ TypeScript에선?

```
// in TypeScript  
  
import styled from 'styled-components';  
import { MenuBarProps, Button } from '../types.ts';  
  
export const MenuBar = ( color, isGap : MenuBarProps ) => {  
  <Button $color={color} $isGap={isGap}>버튼</Button>  
}  
  
const Button = styled.button<Button>`  
  ...  
  color: ${({ $color }) => $color || '#000000' };  
  gap: ${({ $isGap }) => $isGap ? '10px' : 0 };  
  ...  
`;  
;
```

```
// types.ts  
  
export interface Button {  
  $color: string;  
  $isGap: boolean;  
}
```

## 피그마보고 코드 설계하기



피그마에 적힌 style 코드를 그대로 따라 적으면 안 된다는 건 다들 알고 있겠죠? 디자인과 해당 컴포넌트의 쓰임을 살펴보고 이 코드가 정말 **디자이너가 의도한 게 맞는지**를 생각해봐야합니다 🤔

## ▼ 같이 해봐요 🙌

### 1. GlobalStyle을 설정해주자

styles 폴더 아래에 `GlobalStyles.ts` 파일을 만듭니다.

**Reset CSS**를 해줘야 합니다.

브라우저마다 가지고 있는 기본 스타일 값을 초기화해주는 과정으로 모두 동일한 디자인을 화면에서 볼 수 있게 합니다.

`styled-reset` 이란 스타일 초기화 패키지도 있지만 **Reset CSS** 속 코드를 이용하면 추가 패키지 설치 없이도 충분히 기본값을 초기화 시켜줄 수 있습니다.

styled-components로 전역 스타일을 설정하는 법 다들 기억하시나요? 🙄

### 2. GlobalFont를 설정해주자

같은 폴더에 `GlobalFonts.ts` 파일을 만듭니다.

같은 폰트라도 weight를 다르게 적용해야하는 경우 weight별로 `@font-face` 를 추가 해줘야 합니다.

폰트는 여기 있어요!

### 3. Setting한 GlobalStyle을 적용해주자

App.tsx로 돌아와서 그 다음은 ? 🙄

### 4. Header 컴포넌트 만들기

모든 세팅이 끝났다면 커밋 한 번 찍어주시고!

같이 피그마 보면서 Header를 만들어보아요 😊

## ▼ Code

```
// GlobalStyles.ts

import { createGlobalStyle } from 'styled-components';

export const GlobalStyles = createGlobalStyle`

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
  display: block;
}

body {
  line-height: 1;
}
```

```

ol, ul {
  list-style: none;
}

blockquote, q {
  quotes: none;
}

blockquote:before, blockquote:after,
q:before, q:after {
  content: '';
  content: none;
}

table {
  border-collapse: collapse;
  border-spacing: 0;
}

*,
*::before,
*::after {
  box-sizing: border-box;
}

button{
  all: unset;
  cursor: pointer;
}

*,body {
  font-family: 'Pretendard';
}

`;

```

```

// GlobalFonts.ts

import { createGlobalStyle } from 'styled-components';

export const GlobalFonts = createGlobalStyle`

@font-face {
  font-family: 'Pretendard';
  src: url('https://cdn.jsdelivr.net/gh/Project-Noonnu/noonfonts_2107@1.1/Pretendard-Regular.woff') format('woff');
  font-weight: 400;
  font-style: normal;
}

@font-face {
  font-family: 'Pretendard';
  src: url('https://cdn.jsdelivr.net/gh/Project-Noonnu/noonfonts_2107@1.1/Pretendard-Bold.woff') format('woff');
  font-weight: 700;
  font-style: normal;
}

`;

```

```

// Tab.tsx

import styled from 'styled-components';
import { PropsWithChildren } from 'react';

interface TabProps {
  isActive?: boolean;
}

export const Tab = ({
  isActive = false,
  children,
}: PropsWithChildren<TabProps>) => {

```

```

    return <Button $isActive={isActive}>{children}</Button>;
  };

const Button = styled.button<{ $isActive: boolean }>`
  padding: ${({ $isActive }) => ($isActive ? '22px 0 18px 0' : '22px 0')};
  border-bottom: ${({ $isActive, theme }) =>
    $isActive && `4px solid ${theme.color.green}`};
  ${({ theme }) => theme.typography.title1};
  color: ${({ $isActive, theme }) =>
    $isActive ? theme.color.green : theme.color.gray1};

  transition: all 0.2s ease-in-out;

  &:hover {
    padding: 22px 0 18px 0;
    color: ${({ theme }) => theme.color.green};
    border-bottom: 4px solid ${({ theme }) => theme.color.green};
  }
`;

```

```

// TabBar.tsx

import styled from 'styled-components';
import { Tab } from './Tab';

export interface Tab {
  id: number;
  title: string;
}

interface TabBarProps {
  tabs: Tab[];
}

export const TabBar = ({ tabs }: TabBarProps) => {
  return (
    <Container>
      {tabs.map((tab) => (
        <Tab key={tab.id}>{tab.title}</Tab>
      ))}
    </Container>
  );
};

const Container = styled.div`
  display: flex;
  gap: 40px;
`;

```

```

// Header.tsx

import styled from 'styled-components';
import { Tab, TabBar } from './TabBar';
import { ReactComponent as LogoSvg } from '../icons/Lion.svg';

interface HeaderProps {
  onClickLogo: () => void;
  tabs: Tab[];
}

export const Header = ({ onClickLogo, tabs }: HeaderProps) => {
  return (
    <Container>
      <InnerContainer>
        <Logo onClick={onClickLogo} />
        <TabBar tabs={tabs} />
      </InnerContainer>
    </Container>
  );
};

```



```
const Container = styled.header`
  display: flex;
  justify-content: center;
  height: 70px;
  border-bottom: 1px solid ${({ theme }) => theme.color.gray3};
  position: sticky;
  background-color: #fff;
  top: 0;
`;

const InnerContainer = styled.div`
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 0 20px;
  max-width: 1200px;
  width: 100%;
`;

const Logo = styled(LogoSvg)`
  cursor: pointer;
`;
```

```
// App.tsx

import { useNavigate } from 'react-router-dom';
import { ThemeProvider } from 'styled-components';
import { theme } from '../ds/theme';
import { GlobalStyles } from '../styles/GlobalStyles';
import { GlobalFonts } from '../styles/GlobalFonts';
import { Header } from '../ds/components/Header';

function App() {
  const navigate = useNavigate();

  const tabs = [
    {
      id: 0,
      title: '로그인',
    },
    {
      id: 1,
      title: '회원가입',
    },
  ];

  return (
    <ThemeProvider theme={theme}>
      <GlobalStyles />
      <GlobalFonts />
      <Header onClickLogo={() => navigate('/')}> tabs={tabs} />
      ...
    </ThemeProvider>
  );
}

export default App;
```

## 과제 - 오늘 실습에서 끝내지 못한 디자인 적용

컴포넌트 하나하나 설계의 정석대로 구현하려면 생각보다 쉽지 않을 수 있어요.

다음 주차 세션 전까지 이번 세션 때 함께 한 피그마 읽는 법을 기반으로 컴포넌트 설계와 더불어 페이지 디자인을 완성해보세요!

그리고 꼭 **코드 리뷰**를 받아보시는 걸 추천드립니다. (특히 Input 컴포넌트)

원래는 컴포넌트 하나당 브랜치를 생성해 작업하는 것도 좋은 방법이지만 스타일 적용을 위한 브랜치를 하나 새로 분기해 작업 후, 풀리퀘스트를 오픈해 코드 리뷰를 받아보는 방식도 추천드립니다 :)

모두 고생하셨습니다 🥰