2주차 발표자료

1. CSS 모듈과 Sass

CSS Modules이란?

CSS 모듈은 CSS 작성 방식의 하나로, CSS 클래스 이름의 지역 범위를 자동으로 지정하여 클래스 이름 충돌을 방지하는 기술입니다

CSS 모듈이 왜 필요한데?

일반적인 css를 사용하면 어떤 문제가 생기는지 다음 코드를 봅시다.

```
// style.css
.button {
  background-color: blue;
  color: white;
  padding: 10px 20px;
}

// style2.css
.button {
    background-color: red;
    color: black;
}
```

```
<!-- index.html -->
<button class="button">일반 버튼</button>
<button class="button">또 다른 버튼</button>
```

여기서 `.button` 클래스는 **전역 범위**를 가집니다. 즉, 웹 페이지 내의 모든 `.button` 클래스를 가진 요소에 동일한 스타일이 적용됩니다. 만약 위와 같이 style.css뿐만 아니라 style2.css에서도 `.button` 클래스를 정의하면 가장 마지막에 선언된 스타일이 덮어쓰이게 되기 때문에 예상치 못한 결과를 초래합니다.

css module을 사용하면 어떻게 해결할 수 있는지 다음 코드를 봅시다.

```css
// button.module.css
.button {
  background-color: green;
  color: white;
  padding: 10px 20px;
}
```

```jsx
import styles from './button.module.css';

function Button({ children }) {
  return (
    <button className={styles.button}>
      {children}
    </button>
  );
}

export default Button;
```

```html
<!-- 최종적으로 렌더링된 index.html -->
<button class="button_button__randomHash1">모듈 버튼</button>
```

CSS 모듈을 사용하면 `.button` 클래스는 **지역 범위**를 가집니다. JavaScript 파일에서 `styles.button`으로 접근하면, 실제 HTML에서는 브라우저가 이해할 수 있는 고유한 클래스 이름 `button_button__randomHash1` 등 으로 변환됩니다. 이렇게 함으로써 앱의 규모가 커질수록 CSS 클래스 이름이 겹치게 될 가능성을 없애줍니다.그렇기 때문에 다른 컴포넌트나 CSS 파일에서 같은 이름의 클래스를 사용하더라도 **스타일 충돌이 일어나지 않습니다.**

2. Sass란?

Sass(Syntactically Awesome Style Sheets)는 CSS 전처리기(CSS Preprocessor)입니다. CSS의 기능을 확장하여 변수, 중첩 규칙, 믹스인, 상속 등 프로그래밍 언어와 유사한 기능을 제공함으로써 CSS 작성을 더욱 효율적이고 유지보수하기 쉽게 만들어줍니다.

Sass가 왜 필요한데?

순수한 CSS만으로는 복잡하고 반복적인 스타일을 관리하기 어렵습니다. Sass를 사용하면 다음과 같은 장점을 통해 이러한 어려움을 해소할 수 있습니다.

- **변수(Variables):** 색상, 폰트 크기, 간격 등 자주 사용되는 값을 변수로 정의하여 일관성을 유지하고 수정 작업을 용이하게 합니다.SCSS

```scss
$primary-color: blue;
$font-size-large: 18px;

.title {
  color: $primary-color;
  font-size: $font-size-large;
}

.button {
  background-color: $primary-color;
  font-size: 16px;
}
```

- **중첩 규칙(Nesting):** HTML 구조와 유사하게 CSS 규칙을 중첩하여 작성할 수 있어 코드의 가독성을 높이고 특정 요소에 대한 스타일을 명확하게 표현할 수 있습니다.SCSS

```scss
.container {
  width: 100%;
  margin: 0 auto;

  .item {
    font-size: 14px;

    a {
      color: green;

      &:hover {
        text-decoration: underline;
      }
```

```
    }
  }
}
```

- **믹스인(Mixins):** 재사용 가능한 CSS 선언 그룹을 정의하여 여러 스타일 규칙에 쉽게 포함시킬 수 있습니다. 벤더 프리픽스, 공통 스타일 등을 믹스인으로 만들어 생산성을 향상시킬 수 있습니다.SCSS

```scss
@mixin rounded-corners($radius) {
  border-radius: $radius;
  -webkit-border-radius: $radius;
  -moz-border-radius: $radius;
}

.button {
  @include rounded-corners(5px);
  border: 1px solid #ccc;
  padding: 10px;
}

.alert {
  @include rounded-corners(10px);
  background-color: yellow;
  padding: 15px;
}
```

- **상속/확장(Extend/Inheritance):** 다른 스타일 규칙의 속성을 상속받아 코드의 중복을 줄이고 유지보수성을 높일 수 있습니다.SCSS

```scss
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}
```

```
.success {
  @extend .message;
  background-color: lightgreen;
  border-color: green;
}

.error {
  @extend .message;
  background-color: lightcoral;
  border-color: red;
  color: white;
}
```

- **연산(Operators):** CSS 속성 값에 대한 간단한 산술 연산을 수행할 수 있어 동적인 스타일 계산을 용이하게 합니다.SCSS

```
.container {
  width: 100% / 3 * 2; // 66.66%
  padding: 10px * 2;   // 20px
  margin-bottom: 10px + 5px; // 15px
}
```

Sass 사용 방법

Sass 파일(`.scss` 또는 `.sass` 확장자)을 작성한 후, Sass 컴파일러를 사용하여 일반 CSS 파일(`.css` 확장자)로 변환해야 웹 브라우저에서 해당 스타일을 인식할 수 있습니다. Sass 컴파일러는 Node.js 기반의 `sass` 패키지, Ruby 기반의 `sass` gem 등 다양한 도구를 통해 사용할 수 있습니다. 웹팩(Webpack), Parcel과 같은 번들러를 사용하는 경우 Sass 로더를 설정하여 빌드 과정에서 자동으로 CSS 파일로 컴파일하도록 구성할 수 있습니다.

3. CSS 모듈과 Sass 함께 사용하기

CSS 모듈과 Sass는 서로 보완적인 기술로, 함께 사용하여 CSS 개발 경험을 더욱 향상시킬 수 있습니다.

- **Sass의 강력한 기능 활용:** Sass의 변수, 중첩, 믹스인 등의 기능을 CSS 모듈 파일(`.module.scss` 등) 내에서 사용하여 스타일을 더욱 체계적이고 효율적으로 작성할 수 있

습니다.SCSS

```
// button.module.scss
$primary-color: blue;
$padding: 10px 20px;

.button {
  background-color: $primary-color;
  color: white;
  padding: $padding;

  &:hover {
    background-color: darken($primary-color, 10%);
  }
}
```

- **CSS 모듈로 이름 충돌 방지:** Sass로 작성된 스타일을 CSS 모듈로 가져와 사용함으로써 클래스 이름 충돌 문제를 근본적으로 해결할 수 있습니다.JavaScript

```
import styles from './button.module.scss';

function Button({ children }) {
  return (
    <button className={styles.button}>
      {children}
    </button>
  );
}

export default Button;
```

결론적으로, CSS 모듈은 CSS 클래스 이름의 지역 범위를 제공하여 유지보수가 용이하고 예측 가능한 스타일을 만들 수 있도록 돕고, Sass는 CSS 작성 방식을 확장하여 생산성과 코드의 재사용성을 높여줍니다. 이 두 가지 기술을 함께 사용하면 대규모 웹 애플리케이션의 스타일링을 효과적으로 관리할 수 있습니다.

2. Tailwind CSS 통합

Tailwind란?

tailwind는 CSS 프레임워크로써 HTML 요소에 빠르고 간편하게 스타일을 적용할 수 있도록 도와주는 도구

코드로 이해해봅시다. 💻

일반 CSS

```
.my-text {
  color: blue;
  background-color: black;
  padding: 10px;
  border-radius: 5px;
}
```

```
<div class="my-text">안녕하세요!</div>
```

Tailwind

```
<div class="text-blue-500 bg-black p-2 rounded-md">안녕하세요!</div>
```

- **일반 CSS:** 의미 있는 클래스 이름을 직접 정의하고, CSS 파일에서 해당 클래스에 대한 스타일 규칙을 작성해야 합니다.
- **Tailwind CSS:** `text-blue-500` (파란색 글자), `black` (검정색), `p-2` (8px 패딩), `rounded-md` (중간 둥근 모서리) 와 같이 **스타일 속성 자체를 나타내는 직관적인 클래스 이름**을 HTML 요소에 직접 적용합니다.

장점

- **빠른 개발 속도:** 미리 정의된 유틸리티 클래스를 사용하면 CSS 파일을 직접 작성하는 시간과 노력을 줄여서 훨씬 빠르게 웹 페이지를 스타일링할 수 있습니다. 간단한 스타일 변경은 HTML 파일에서 바로 적용할 수 있어서 작업 효율성이 높아져요

- **작은 번들 크기:** Production 빌드 시 사용하지 않는 CSS 규칙을 자동으로 제거해 줍니다. 따라서 최종 CSS 파일의 크기를 최소화하여 웹 페이지의 로딩 속도를 향상시키는 데 도움이 됩니다.

단점

- **HTML의 장황함:** 많은 스타일이 HTML 클래스에 직접 추가되기 때문에 HTML 코드가 길어지고 복잡해 보일 수 있습니다. 특히 복잡한 UI 컴포넌트의 경우, 수많은 클래스가 HTML 요소에 나열될 수 있습니다.
- **추상화 부족:** CSS 속성을 직접적으로 드러내는 유틸리티 클래스 위주로 작성되기 때문에, 의미론적(semantic) 클래스 이름을 사용하는 것에 비해 스타일의 추상화 정도가 낮아질 수 있습니다.

Tailwind 적용법
1. Next.js 문서:
https://nextjs.org/docs/app/building-your-application/styling/tailwind-css

2.Tailwind 문서: https://tailwindcss.com/docs/installation/framework-guides/nextjs

3. CSS-in-JS 솔루션

CSS-in-JS는 자바스크립트 파일 내에서 CSS를 작성하고 관리하는 방식입니다. 이는 컴포넌트 기반 개발 환경에서 스타일링의 효율성과 유지보수성을 높이기 위해 등장했습니다.

CSS-in-JS가 왜 필요한데?

전통적인 CSS 방식은 전역 스코프를 가지기 때문에 프로젝트 규모가 커질수록 다음과 같은 문제점을 야기할 수 있습니다.

- **스타일 충돌:** 여러 CSS 파일에서 동일한 클래스 이름을 사용할 경우, 예상치 못한 스타일이 적용될 수 있습니다.
- **명명 규칙의 어려움:** 클래스 이름이 전역적으로 유일해야 하므로, 길고 복잡한 명명 규칙을 따르거나 접두사를 사용하는 등의 번거로움이 있습니다.
- **사용하지 않는 스타일 관리의 복잡성:** 특정 컴포넌트에서만 사용되는 스타일을 분리하고 관리하기 어렵고, 더 이상 사용하지 않는 스타일을 추적하여 제거하는 것이 번거롭습니다.

다음 코드를 통해 일반적인 CSS 사용 시 발생할 수 있는 문제점을 살펴보겠습니다.

CSS

```css
/* style.css */
.button {
  background-color: blue;
  color: white;
  padding: 10px 20px;
}
```

CSS

```css
/* style2.css */
.button {
    background-color: red;
    color: black;
}
```

HTML

```html
<button class="button">일반 버튼</button>
<button class="button">또 다른 버튼</button>
```

위 예시에서 `.button` 클래스는 전역 범위를 가지므로, `style2.css`에서 정의된 스타일이 `style.css`의 스타일을 덮어쓰게 되어 두 버튼 모두 빨간색 배경에 검은색 글씨로 표시됩니다. 이는 개발자의 의도와 다를 수 있으며, 프로젝트 규모가 커질수록 이러한 스타일 충돌의 가능성은 더욱 증가합니다.

CSS-in-JS를 사용하면 어떻게 해결할 수 있을까?

CSS-in-JS 솔루션은 스타일을 자바스크립트 컴포넌트와 함께 정의하고, 각 스타일이 해당 컴포넌트의 스코프 내에서만 적용되도록 합니다. 이를 통해 클래스 이름 충돌을 방지하고, 컴포넌트와 스타일 간의 의존성을 명확하게 관리할 수 있습니다.

다음은 CSS-in-JS의 한 형태인 Styled Components를 사용한 코드 예시 입니다.

JavaScript

```
import styled from 'styled-components';

const StyledButton = styled.button`
  background-color: green;
  color: white;
  padding: 10px 20px;
`;

function MyButton({ children }) {
  return (
    <StyledButton>
      {children}
    </StyledButton>
  );
}

export default MyButton;
```

위 코드에서 `styled.button`을 사용하여 만든 `StyledButton`은 해당 컴포넌트 내에서만 유효한 스타일을 가지게 됩니다. Styled Components는 런타임 시 고유한 클래스 이름을 생성하여 스타일을 적용하므로, 다른 컴포넌트나 CSS 파일에서 동일한 클래스 이름을 사용하더라도 스타일 충돌이 발생하지 않습니다.

최종적으로 렌더링된 HTML은 다음과 유사한 형태가 됩니다.

HTML

```
<button class="sc-dkPtRN hxYfvd">모듈 버튼</button>
```

`sc-dkPtRN`과 `hxYfvd`는 Styled Components가 자동으로 생성한 고유한 클래스 이름입니다. 이를 통해 스타일이 특정 컴포넌트에만 적용되도록 보장합니다.

이처럼 CSS-in-JS는 스타일을 컴포넌트와 함께 관리함으로써 스타일 충돌을 방지하고, 코드의 응집성을 높여 유지보수를 용이하게 합니다. 또한, 자바스크립트의 기능을 활용하여 동적인 스타일링을 더욱 효과적으로 처리할 수 있다는 장점을 가집니다.