ChatGPT

성숙한 디자인 시스템의 디자인 토큰 체계 분석

디자인 토큰은 웹, iOS, 안드로이드 등 **플랫폼을 초월**하여 색상, 글꼴, 간격 등 스타일 값을 일관되게 관리하는 **공통 언어**입니다 [1]. Netflix, Uber Base, Google Material Design, Salesforce Lightning와 같은 성숙한 디자인 시스템들은 이러한 토큰을 활용해 여러 제품과 테마 간에 **일관성**과 **확장성**을 유지하고 있습니다. 디자인 토큰은 일반적으로 JSON 등의 포맷으로 정의되어 각 플랫폼 코드(CSS, iOS, Android 등)로 **자동 변환**되며 [1], **브랜드 변경이나 다크모드**와 같은 테마 전환도 토큰 값만 교체하면 전체 UI에 반영되도록 설계됩니다 [2].

디자인 토큰은 계층적으로 관리되며 흔히 **원시 토큰(primitive) – 의미 토큰(semantic) – 컴포넌트 토큰(component)**의 구조를 갖춥니다 [3] [4]. 원시 토큰은 색상 값, 폰트 크기, 간격 수치 등 **기본 값**(예: HEX 색상, px 단위)들을 정의하고 [5], 의미 토큰은 이러한 값에 **사용 목적의 의미**를 부여한 추상화(예: `color.primary`, `spacing.small`)로서 주로 원시 토큰을 참조합니다 [4]. 컴포넌트 토큰은 개별 구성요소에 특화된 토큰으로, 보통 의미 토큰을 참조하여 버튼이나 입력 필드 등 **특정 컴포넌트의 스타일 결정**에 쓰입니다 [6]. 이러한 계층화로 디자인 변경 시 **토큰 한 곳만 수정하면 전체 반영**되며, 다크 모드나 멀티 브랜드 지원 시에도 토큰 세트만 교체하면 UI 전반을 쉽게 바꿀 수 있습니다 [2].

아래에서는 **Color, Typography, Spacing, Radius, Shadow, Border, Motion, State** 항목별로 토큰 체계 구조와 예시를 정리합니다. 각 항목마다 **원시 vs. 의미 토큰 분류**, **플랫폼 불문 통합 관리 방식(JSON 등)**, **브랜드 확장성과 유지보수** 관점을 설명하며, 실제 토큰 명명 규칙과 값 정의, 사용 전략 사례를 함께 제공합니다.

Color (색상)

구조와 계층: 색상 토큰은 보통 **팔레트 기반의 원시 색상**과 **UI 의미에 따른 추상적 색상**의 이중 구조로 정의됩니다 [7]. 우선 디지털 제품에서 사용할 색상 팔레트를 정의한 뒤, 이를 용도에 따라 의미 있는 이름으로 할당합니다. 예를 들어 **원시 토큰**으로 `palette.blue.500`, `palette.gray.100` 등의 색상 값(HEX/RGB)을 정의하고, **의미 토큰**으로 `color.brand.primary`, `color.text.default`, `color.background.subtle` 등을 두어 팔레트 색상을 참조하도록 합니다 [7] [8]. 이렇게 하면 팔레트의 특정 블루 계열 색상이 브랜드 기본색으로 쓰일 경우 `color.brand.primary`가 그 값을 가리키며, 디자이너와 개발자는 **"primary 브랜드 색"** 토큰만 참조하면 되므로 의미가 명확해집니다 [9].

- **원시(Color Palette) 토큰:** 브랜드 팔레트 색상 자체를 정의. 예) `palette.red.500 = #E53935`
- **의미(Semantic) 토큰:** 사용 목적에 따른 색상. 예) `color.background.default` (기본 배경색) = 팔레트 중 밝은 회색 [8], `color.text.primary` (본문 기본 글자색) = 팔레트 중 어두운색 [10].
- **컴포넌트 토큰:** 특정 컴포넌트 전용 색. 필요 시 사용. 예) `button.primary.background = color.brand.primary` 등.

플랫폼 독립성: 색상 토큰은 일반적으로 **JSON 등의 중립적 형식**으로 저장되고, 빌드 도구를 통해 SCSS 변수, CSS Custom Property, Android XML, iOS Swift 코드 등으로 변환됩니다 [1]. 예를 들어 `color.text.primary` 토큰이 JSON에 `#333333`으로 정의되면, 웹에서는 `--color-text-primary: #333333;` CSS 변수로, iOS에서는 UIColor 값으로 자동 매핑됩니다. 토큰 이름에는 **카테고리 접두사**(`color`)를 붙여 한눈에 종류를 알 수 있게 하며 [11], 값 타입도 메타데이터로 명시하여 (예: color 타입) 각 플랫폼에 맞게 처리합니다 [11]. 이렇게 함으로써 **동일 토큰 세트로 iOS/Android/Web 디자인 동기화**를 이룹니다.

브랜드 확장성과 유지보수: 색상 토큰 체계는 **다크 모드**나 **여러 브랜드 테마**를 손쉽게 지원하도록 설계됩니다. 의미 토큰을 사용하면 예컨대 `color.brand.primary`의 값만 교체하여 **전체 제품의 주색상을 변경** 할 수 있습니다 [12]. Salesforce Lightning에서는 토큰 이름에 `BRAND` 접두사를 붙인 토큰들을 **브랜드 가변 토큰**으로 지정하며, 고객 조직의 테마 설정에 따라 이 값들이 변경되도록 했습니다 [13]. 예를 들어 `$brand-accessible` 토큰은 기본 브랜드 색상의 접근성 준수 변형인데, 브랜드 팔레트를 바꾸면 이 토큰 값도 새 팔레트에 맞게 바뀌어 UI 전반의 색이 일관되게 변합니다 [13] [14]. 또한 Uber의 **Base Design System**처럼 **색상 계층 구조**를 활용하는 사례도 있습니다. Uber는 중립색(gray)을 용도에 따라 배경용, 콘텐츠(텍스트)용, 테두리용 세 카테고리로 나누고 `color-background-*`, `color-content-*`, `color-border-*` 형태의 토큰을 제공합니다 [15] [8]. 예를 들어 `color-border-subtle`이라는 토큰은 은은한 경계선에 사용하는 회색 계열 색으로 정의되어, 디자이너가 **"보더용 미묘한 회색"**을 일관되게 쓸 수 있습니다 [8]. 이렇듯 의미 계층을 도입하면 **색상 사용 가이드**를 내재화하여, 팔레트에서 임의 색을 쓰지 않고 정해진 토큰만 쓰게 함으로써 유지보수성을 높입니다 [15].

토큰 명명 규칙과 예시: 색상 토큰은 일반적으로 `color用途.세부용도.상태` 형태로 명명합니다. 예컨대 `color.button.primary.hover`는 "버튼/Primary버전/호버 상태"의 색상을 의미합니다 [16] [17]. 이렇게 이름만 보고도 **역할과 상황**을 알 수 있게 하여, 디자이너와 개발자가 토큰 용도를 직관적으로 이해하도록 합니다 [18] [19]. 반면 팔레트 기반 원시 토큰은 `color.blue.50`처럼 색상+농도 번호로 정의되거나 [20], `$palette-blue-30: #014486`처럼 팔레트 이름으로 관리되기도 합니다 [21] [22]. 중요한 것은 **의미 토큰이 팔레트에 간접 참조** 되어, 팔레트 값이 바뀌어도 의미 토큰 이름(`primary`, `success` 등)은 그대로 쓰면서 내부 값만 변경되도록 하는 것입니다 [4]. 이로써 제품의 브랜드 색 변경이나 다크모드 전환 시 **토큰 값 교체만으로 UI를 일괄 업데이트**할 수 있습니다 [2].

Typography (타이포그래피)

구조와 계층: 타이포그래피 토큰은 **폰트 패밀리, 폰트 크기, 글줄 높이(line-height), 글꼴 두께(font-weight)** 등을 정의하며, 보통 **타입 스케일**을 원시 토큰으로, **텍스트 스타일 이름**을 의미 토큰으로 관리합니다. 예를 들어 Salesforce Lightning에서는 폰트 크기를 1~11까지 **단계적 스케일** 토큰으로 정의하고 (`$font-size-1` ~ `$font-size-11`), 각 단계별 px값을 지정합니다 [23] [24]. 그러고 나서 이를 활용해 **의미 있는 이름**의 토큰이나 스타일을 정의하는데, 예를 들어 `$font-size-5` (16px)는 본문(body) 기본 크기로, `$font-size-7` (20px)은 헤드라인 소제목 크기 등으로 약속합니다. 실제 Lightning 시스템에서는 `$form-label-font-size` 같은 토큰을 두어 **양식 레이블 폰트 크기**를 정의했고, 그 값으로 `$font-size-2` (12px)를 참조하도록 했습니다 [25]. 이처럼 **원시 크기 토큰 → 의미 토큰**의 계층을 두면, 나중에 전체 타입 스케일을 키우거나 줄일 때 원시 토큰 값만 수정하면 여러 스타일에 일괄 반영됩니다. 폰트 두께 역시 `$font-weight-regular`=400, `$font-weight-bold`=700 등을 토큰화하여 전체에서 통일해서 사용합니다 [26] [27].

- **원시 토큰:** 폰트 관련 기본 값들. 예) `font-size-5 = 16px` [28], `font-weight-bold = 700` [26], `font-family-default` = "Roboto, Arial, sans-serif" 등 [29].
- **의미 토큰:** 사용 맥락에 이름 붙인 토큰. 예) `typography.body.small = font-size-4 + regular` (작은 본문 스타일), `typography.heading.large = font-size-8 + bold` 등. Material Design 3의 경우 **스타일 이름** 자체를 토큰으로 하여 *Title Large*, *Body Medium* 등의 텍스트 스타일을 정의하고 각 스타일에 폰트 패밀리/크기/두께/자간 등을 맵핑합니다.
- **컴포넌트 토큰:** 컴포넌트 전용 폰트 속성. 예) `button.label.font-size = font-size-3` 등 특정 컴포넌트에 특화된 경우.

플랫폼 독립성: 글꼴 토큰도 디자인 토큰 툴(chain)을 통해 iOS/Android 플랫폼의 단위로 변환됩니다. 예를 들어 웹에서는 `1rem`으로 정의된 폰트 크기 토큰이 Android에서는 `sp`로, iOS에서는 `pt`로 변환되는 식입니다. **기준 단위**를 활용하는 것도 특징인데, 웹에선 모든 폰트 크기를 `rem`나 `em`으로 출력하여 사용자 환경별 **반응형 확대**를 지원하고 [30], iOS/

안드로이드에선 OS의 접근성 설정 (폰트 사이즈 조절)에 대응하도록 합니다. 토큰 저장은 JSON에서 아래와 같이 가능하며, 변환 시 단위 접미사를 붙입니다:

```
{
  "fontSizes": {
    "1": { "value": 10, "type": "fontSize" },
    "2": { "value": 12, "type": "fontSize" },
    ...
  },
  "fontWeights": {
    "regular": { "value": 400 },
    "bold": { "value": 700 }
  }
}
```

위 JSON을 통해 각 플랫폼 코드에서 `font-size-2`는 `12px` 또는 `12sp` 등으로 적용되고, `font-weight-bold`는 CSS에서는 `700`, iOS에서는 UIFontWeight.semibold 등으로 대응시킬 수 있습니다.

브랜드 확장성과 유지보수: 타이포그래피 토큰은 **브랜드 개성**을 반영하고 변경하기 쉽도록 구조화됩니다. 예를 들어 **브랜드 서체 변경**이 필요한 경우, `font-family` 토큰 값만 교체하면 전체 컴포넌트에 적용된 폰트가 일괄 변경됩니다 [29]. 다크 모드의 경우 일반적으로 동일 폰트를 쓰므로 색상처럼 교체 필요는 없지만, **다른 브랜드 간 서체나 크기 규모(scale)**가 다를 때 토큰 체계로 유연하게 대처 가능합니다. 한 브랜드에서는 `$font-size-5=16px`로 기본 본문 크기를 쓰다가, 다른 브랜드나 플랫폼(예: TV, 웨어러블 등)은 더 크거나 작은 값으로 조정해야 하면 해당 토큰 세트를 분리된 **테마별 JSON**으로 관리하고 빌드시 선택하여 적용할 수 있습니다 [6]. 또한 토큰으로 관리하면 **접근성** 요구사항 대응도 수월한데, 예를 들어 모든 텍스트에 최소 12px 이상을 쓰도록 해야 할 때 토큰 `font-size-1`을 12px로 정의해두면 실수로 더 작은 글자가 쓰이지 않게 디자인 시스템 차원에서 통제할 수 있습니다.

토큰 명명 규칙과 예시: 타이포그래피 토큰은 보통 카테고리와 규모를 조합해 짓습니다. 예컨대 **Material Design**은 `md.sys.typescale.body.large` 식으로 prefix에 시스템/스타일 정보를 담고 스타일 이름과 크기 구분을 합니다. 일반적으로 디자인 시스템에서는 **텍스트 스타일 이름** (예: Heading, Body, Caption 등) + **크기나 중요도 등급** (Large, Small 등)으로 토큰을 나눠 정의합니다. Uber Base 디자인 시스템에서도 글꼴 크기/두께 토큰을 사용하며, **Typography Scale**을 정해서 `font-size-scale-100, 200...` 등 숫자로 표기하거나 Lightning처럼 `font-size-1...11` 등으로 나열합니다 [23] [31]. 그런 다음 의미 단위로 폰트크기+두께 조합을 만들어 `typography.heading1 = size-8 + bold` 식으로 운용합니다. 예를 들어 **Salesforce Lightning**에서는 `$font-weight-bold`/`regular`를 정의하고 `$page-header-title-font-weight = bold` 식으로 특정 사례에 적용했으며 [32], `$tabs-font-weight = bold` 등 컴포넌트별 토큰도 존재합니다 [32] [33]. 이처럼 **공통 폰트 토큰** + **문맥별 토큰** 구조로 관리하면, 폰트 두께나 크기 변경 시 한곳의 토큰 값으로 여러 요소를 업데이트할 수 있어 유지보수에 용이합니다.

Spacing (간격)

구조와 계층: Spacing 토큰(여백/간격 토큰)은 UI에서 사용하는 **여백, 패딩, 마진 간격**들을 정의합니다. 일반적으로 **크기 스케일**을 원시 토큰으로 만들어 몇 가지 표준간격을 제공하고, 필요에 따라 의미 토큰이나 컴포넌트 토큰으로 활용합니다. 예를 들어 많은 디자인 시스템이 **4px 또는 8px 그리드**를 기반으로 간격을 설정하는데, Lightning Design System에서는

`$spacing-xxx-small = 2px`, `$spacing-xx-small = 4px`, `$spacing-x-small = 8px`, `$spacing-small = 12px`, `$spacing-medium = 16px` … `$spacing-xx-large = 48px` 식으로 **일정 배수 간격** 토큰을 제공합니다 [34] [35]. 이러한 원시 토큰을 사용하여 컴포넌트나 레이아웃에 의미 있게 적용합니다. 예를 들어 카드 컴포넌트의 내부 패딩을 `spacing-medium (16px)`으로 규격화하면, 나중에 전체 디자인의 공간감조정을 위해 `spacing-medium` 값을 한꺼번에 조절할 수 있습니다. 일부 시스템에서는 **세로/가로 방향**을 구분한 토큰(`spacing.horizontal.small` 등)이나 **가변 간격 토큰**(responsive spacing)도 정의하여 레이아웃의 유연성을 높입니다 [36] [37].

- **원시 토큰:** 기본 간격 값. 예) `spacing-1 = 4px`, `spacing-small = 8px` [35], `spacing-medium = 16px` [38], `spacing-large = 24px` 등. 각 값은 4px 단위 등 일정 비율로 증가하도록 설계합니다.
- **의미 토큰:** (필요한 경우) 맥락을 부여한 간격. 예) `spacing.form.fieldGap = spacing-medium` 등 폼 필드 사이 간격을 따로 토큰화할 수 있습니다. 일반적으로는 간격은 수치 자체로 충분히 의미를 가져 별도 의미 이름을 잘 안 붙이지만, **컴포넌트 특정 패턴** (예: 표 셀 간격 `table.cellSpacing = 4px 8px`)을 토큰으로 갖기도 합니다 [39] [40].
- **컴포넌트 토큰:** 컴포넌트 전용 여백. 예) `button.padding.horizontal = spacing-small (8px)` 등 버튼 좌우패딩을 정의. Lightning에서는 `$table-cell-spacing = 4px 8px` 같이 특정 컴포넌트 조합값도 토큰으로 저장했습니다 [39].

플랫폼 독립성: 간격 토큰의 값은 주로 px/rem 등의 절대/상대단위로 정의되며, 플랫폼별로 환산됩니다. 예를 들어 JSON에 `spacingSmall: { value: 8, unit: "pixel" }`가 있다면 iOS에서는 8pt, Android에서는 8dp로 해석됩니다. **반응형 디자인**에서는 미디어 쿼리나 컨테이너 크기에 따라 간격을 조절해야 하므로, 간격 토큰을 고정값만 두지 않고 **배수(multiple)**나 **퍼센트**로 정의하기도 합니다. 하지만 일반적으로 디자인 시스템은 **고정된 간격 단계**를 정의하고 반응형 조정은 별도 레이아웃 규칙으로 처리합니다. 토큰은 JSON로 관리되므로 한 번 정의한 간격 값이 모든 플랫폼에서 **일관되게 사용**됩니다. 간혹 플랫폼 특수성에 따라 조정이 필요하면 (예: 모바일에서는 동일 클래스 컴포넌트라도 여백을 더 좁히는 등) 별도 **플랫폼 모드**의 토큰 세트를 가져갈 수 있습니다 [6].

브랜드 확장성과 유지보수: 간격은 브랜드마다 크게 다르지 않을 때가 많지만, **제품 성격**에 따라 여유 있는 레이아웃 vs. 빽빽한 정보밀도 등의 차이가 있을 수 있습니다. 이럴 때 토큰 세트를 분리하여 **여유형 테마**에서는 spacing 값들을 크게, **조밀형 테마**에서는 작게 운용할 수 있습니다. 디자인 토큰 체계에서는 이러한 테마 간 변경이 쉽도록, 예컨대 `spacing-large` 등의 토큰 이름은 고정하고 값만 테마별 JSON에서 달리하여 적용합니다. 유지보수 면에서, 모든 컴포넌트가 토큰을 참고하도록 하면 **일괄 간격 조정**이 쉬워집니다. 예를 들어 전체 UI의 기본 여백을 8px에서 10px로 바꾸려면 `spacing-base` 토큰 값 하나 수정으로 가능합니다. 반면 토큰 없이 개별 CSS에 값이 흩어져 있었다면 모두 찾아 바꿔야 했을 것입니다 [12].

토큰 명명 규칙과 예시: 간격 토큰은 주로 *spacing* + 크기명으로 명명합니다. Lightning처럼 **직관적 수식어**를 쓰는 경우 `$spacing-small`, `$spacing-medium`, `$spacing-large` 등으로 구분하고 해당 픽셀값을 문서화합니다 [38] [34]. 다른 접근으로 **숫자 등급**을 쓰기도 합니다 (예: `space.1`, `space.2`...). 명명에 일관성을 두면 디자이너가 감각적으로 어떤 크기인지 이해하기 쉬운데, "small, medium, large"는 상대적 의미지만 4px, 8px 배수 체계라는 컨텍스트 안에서는 충분히 유의미합니다. 또한 **축 방향**에 따라 `spacing.horizontal.small`, `spacing.vertical.small`로 나누거나, Apple Human Interface Guidelines처럼 **간격 조합 토큰**(예: 상하/좌우 패딩을 한꺼번에 지정한 토큰)도 생각할 수 있습니다. 예시로, `$template-gutters = 12px` (레이아웃 좌우 여백 토큰) [40] 등이 있습니다. 중요한 것은 **디자인 시스템 전반에 동일 토큰 이름을 사용**하여 어느 컴포넌트에서든 `spacing-medium`이라 하면 동일한 16px 여백으로 해석되게 하는 것입니다 [10].

Radius (모서리 굴곡)

구조와 계층: Radius 토큰은 **모서리 둥글기(테두리 반경)** 값을 정의합니다. 대부분의 디자인 시스템은 2~4개의 반경 값을 표준으로 정해 두고, 컴포넌트 별로 이 중 하나를 적용합니다. 예를 들어 Material Design과 Salesforce Lightning 모두 **Small, Medium, Large** 정도의 반경 토큰을 사용합니다. Lightning의 경우 `$border-radius-small = 2px`, `$border-radius-medium = 4px`, `$border-radius-large = 8px`으로 정의하고, 완전한 원형을 위한 `$border-radius-circle = 50%`도 제공합니다 [41] [42]. 컴포넌트들은 이 토큰을 가져다 써서, 버튼은 small (2px) 모서리, 카드나 모달은 medium (4px) 혹은 large (8px) 모서리 등으로 **일관성**을 유지합니다. 경우에 따라 특정 컴포넌트는 별도 토큰을 갖기도 하는데, Lightning에서는 페이지 헤더용으로 `$page-header-border-radius = 4px (medium)` 토큰을 두어 해당 영역에 적용했습니다 [43].

- **원시 토큰:** 표준 모서리 반경 값. 예) `radius-small = 2px` [44], `radius-medium = 4px`, `radius-large = 8px` [45], `radius-circle = 50%` (정원형).
- **의미 토큰:** (필요시) 특정 용도 이름. 일반적으로 radius는 값 자체로 단순하여 별도 의미 명칭보다는 바로 컴포넌트에 적용합니다.
- **컴포넌트 토큰:** 컴포넌트별 반경. 예) `avatar.borderRadius = radius-circle` 등. 대부분 시스템에서는 굳이 컴포넌트 토큰을 만들기보다 **컴포넌트 스타일 시에 공용 radius 토큰을 선택**하는 방식으로 충분히 관리합니다.

플랫폼 독립성: 반경 토큰도 JSON 등으로 관리되고 px 값을 그대로 플랫폼별 CSS나 코드로 내보냅니다. iOS의 경우 `CALayer.cornerRadius`나 SwiftUI `cornerRadius`에 픽셀과 1:1로 대응되고, Android에서는 `shape` drawable에 dp 단위 반영됩니다. 만약 플랫폼별 렌더링 차이를 고려해야 한다면 (예: 소수점 반경 처리), 변환 단계에서 반올림 등의 처리를 하지만, 일반적으로는 **특별한 단위 변환 없이 동일 수치**를 사용합니다.

브랜드 확장성과 유지보수: 브랜드 성격에 따라 모서리 스타일을 조정하는 경우가 많습니다. 어떤 브랜드는 각지고 샤프한 모서리를, 어떤 곳은 둥글고 친숙한 모서리를 선호합니다. 디자인 토큰으로 반경을 관리하면 테마 변경 시 토큰 값 교체만으로 **전체 UI 분위기를 전환** 할 수 있습니다. 예를 들어 기본 테마에서는 `radius-small=4px`, 라운드형 테마에서는 `radius-small=8px`로 바꾸면 수백 개 컴포넌트의 모서리가 일시에 변경됩니다. 또한 유지보수 관점에서, **일괄 조정**이 용이합니다. 디자인팀 피드백으로 "전체적으로 모서리를 조금만 더 둥글게 하자"는 결정이 내려지면, 토큰 값 2~3개만 수정하면 되는 식입니다.

토큰 명명 규칙과 예시: Radius 토큰은 일반적으로 `radius-크기` 형태로 명명합니다. Lightning처럼 `$border-radius-small/medium/large`로 쓰거나 [45] [46], 짧게 `radius.sm, radius.md, radius.lg`로도 씁니다. 완전한 원형은 `radius.circle` 또는 `radius.round` 등으로 특별 취급합니다 [42]. Uber Base 등 많은 시스템에서 radius 토큰은 값 종류가 많지 않아 명명에 큰 혼동이 없지만, **prefix로 카테고리를 표시**하는 일관성은 유지합니다 (예: borderRadiusSmall vs. borderRadiusLarge 등으로 CamelCase). 명명과 구조보다 중요한 것은 **일관된 사용**으로, 개발자들이 `border-radius: var(--radius-small);`처럼 토큰을 적용하도록 가이드하면 절대 임의의 px값을 쓰지 않게 되어 디자인 시스템의 통제가 강화됩니다.

Shadow (그림자/음영)

구조와 계층: Shadow 토큰은 **그림자 효과(음영)**를 위한 속성들을 캡쳐합니다. CSS의 box-shadow처럼 **x/y 오프셋, 블러, 색상, 투명도** 값의 조합으로 이루어지는데, 디자인 시스템에서는 자주 쓰는 그림자 스타일 몇 개를 토큰화합니다. 두 가지 접근이 있습니다: **단계별 수준(Elevation)**으로 정의하거나, **용도별 그림자**로 정의하는 것입니다. Google Material Design은 Elevation 개념을 도입하여 레벨1~5 등의 높이에 따라 그림자 강도가 달라지도록 했고, 이러한 레벨을 디자인 토큰으로

볼 수 있습니다 (예: `shadow.level1`, `shadow.level2` ...). 반면 Salesforce Lightning은 `$card-shadow`, `$dropdown-shadow` 등 **맥락별 토큰**을 뒀습니다 [47] [48]. Lightning의 경우 `$card-shadow`와 `$page-header-shadow`가 동일 값(0 2px 2px 0 rgba(0,0,0,0.10))으로 정의되어 있는데, 둘 다 작은 떠있는 패널에 쓰는 얕은 그림자 스타일입니다 [47] [49]. 그리고 `$shadow-active`는 인터랙티브 요소가 활성(active)될 때 쓰는 얇은 테두리 형광 효과(파란 그림자)이고 [50] [51], `$shadow-drag`는 드래그앤드롭 시 나타나는 강한 그림자 등으로 구분됩니다 [52]. 이렇게 **상황별로 의미 있는 이름**을 부여하면 해당 상황에 맞는 그림자 외에는 다른 것을 쓰지 않도록 유도할 수 있습니다.

- **원시 토큰:** 그림자 효과의 구성요소 토큰. (많이 사용되진 않음) 예를 들어 일부 시스템은 `shadow-color`, `shadow-offset-small` 등으로 세분화하지만, 일반적으로 그림자는 한꺼번에 문자열로 처리합니다.
- **의미 토큰:** 그림자 유형별 토큰. 예) `shadow.small` (작은 그림자) = `0 1px 2px rgba(0,0,0,0.2)`, `shadow.medium` = `0 2px 4px ...` 등 단계별 정의하거나, `$modal-shadow`, `$popover-shadow` 처럼 컴포넌트별 큰/작은 그림자를 지정하기도 합니다.
- **컴포넌트 토큰:** 특정 컴포넌트 전용 그림자. 대부분 의미 토큰으로 커버 가능하지만, 특수한 경우 컴포넌트 토큰으로 관리합니다 (예: Material의 **pressed state** 강한 그림자 등이 있다면 `button.pressed.shadow` 등).

플랫폼 독립성: 그림자 토큰은 JSON에서 **문자열 형태**로 전체 값을 저장하거나, 몇 개 숫자로 이루어진 배열로 보관되기도 합니다. 그런 다음 웹에서는 CSS `box-shadow`로, iOS는 `CALayer.shadow...` 프로퍼티들로, Android는 elevation이나 MaterialShapeDrawable로 변환됩니다. 다만 CSS의 그림자와 모바일의 그림자 모델이 달라 완전한 1:1 매핑이 어려울 때도 있습니다. 이 경우 디자인 토큰은 주로 웹 중심으로 관리하고, 모바일은 최대한 가까운 효과로 별도 정의하기도 합니다. 예를 들어 iOS의 높은 elevation은 단순 그림자로 표현하기 어려워 **오버레이 투명 레이어** 조합으로 구현하는 등 차이가 생길 수 있습니다. 그래도 **토큰 이름과 의미는 통일**하여, `shadow.level3`이면 모든 플랫폼에서 대략 동일한 깊이감을 주도록 맞춥니다.

브랜드 확장성과 유지보수: 그림자 스타일도 브랜드에 따라 **강조 정도나 색상**이 달라질 수 있습니다. 어떤 브랜드는 뚜렷한 그림자로 입체감을 주고, 어떤 브랜드는 플랫하게 그림자 없이 디자인할 수도 있습니다. 토큰화해 두면 그림자를 쓰지 않는 브랜드에서는 모든 shadow 토큰 값을 `none`이나 투명으로 바꾸거나, 반대로 강조하는 브랜드에서는 더 진한 값으로 바꿀 수 있습니다. 또한 다크 모드에서는 기존 검은 그림자가 잘 안 보이므로 투명도나 색상을 조정해야 하는데, 이 역시 토큰으로 한 곳에서 관리하면 모드 전환 대응이 수월합니다. 유지보수 측면에서는, 예컨대 전체적으로 그림자 블러를 늘려 부드럽게 하고 싶다면 `shadow.medium` 등 토큰 값 한 번 수정으로 일괄 조정 가능합니다.

토큰 명명 규칙과 예시: 그림자 토큰 명명은 주로 `shadow` + 정도/용도로 합니다. **Material Design**처럼 elevation 번호를 쓰는 경우 `shadow-1`, `shadow-2...`로 일관되게 할 수 있습니다. 다른 예로 **Atlassian 디자인 시스템**은 `elevation.surface.raised` 등 맥락있는 이름을 사용합니다. Lightning에서는 `$card-shadow`, `$page-header-shadow`처럼 **컴포넌트 맥락**을 이름에 넣었고 [47] [49], Uber Base는 **Elevation 레벨** 중심으로 관리합니다. 중요한 것은 개발자가 `var(--shadow-small)` 등으로 사용해 **값을 암기하지 않고도 일관된 그림자**를 적용하게 하는 것이며, 토큰명이 용도를 담고 있어야 합니다 (예: `shadow.modal` vs `shadow.button` 등). 아래는 **Salesforce Lightning**의 그림자 토큰 일부 예시입니다:

토큰 이름	의미 (용도)	그림자 값 (`x y blur color`)
`$card-shadow`	카드 컴포넌트 기본 그림자	`0 2px 2px 0 rgba(0, 0, 0, 0.10)` [47]
`$shadow-drop-down`	드롭다운 메뉴용 그림자	`0 2px 3px 0 rgba(0, 0, 0, 0.16)` [48]
`$shadow-active`	인터랙션 활성 상태 강조	`0 0 2px #0176d3` (브랜드 색상의 얇은 글로우) [50]

위에서 보듯 토큰명에 **컴포넌트 또는 상태**를 명시하여 언제 쓰는 그림자인지 알기 쉽게 했습니다. 이처럼 명확한 토큰 체계는 디자이너에게는 재사용 가이드를, 개발자에게는 유지보수의 편의를 제공합니다.

Border (테두리 선)

구조와 계층: Border 관련 토큰은 **테두리 선의 색상, 두께, 스타일** 등을 다룹니다. 색상은 보통 Color 토큰의 일부로 (예: `color.border.default` 등) 다루지만, 디자인 시스템에 따라 별도 **Border Color 카테고리**를 두기도 합니다. 예를 들어 Lightning의 **Border Color** 토큰에는 `$color-border-brand`, `$color-border-input` 등 맥락별 테두리 색들이 정의되어 있습니다 [53] [54]. 두께(thickness)는 흔히 1px, 2px와 같이 소수 몇 개 값만 사용하므로 `$border-width-1 = 1px`, `$border-width-2 = 2px` 등으로 원시 토큰을 정의해두고, `border.thin`, `border.thick`처럼 의미 토큰으로 부를 수도 있습니다. 또한 **스타일**(실선, 점선 등)은 거의 모든 UI에서 실선(solid)만 쓰는 경우가 많아 토큰화하지 않을 때도 있지만, 필요하면 `border.style.default = solid` 등으로 둘 수 있습니다.

정리하면, Border 토큰 체계는 간단한 편이며 다음과 같습니다:

- **원시 토큰:** 테두리 기본 값. 예) `border-width-1 = 1px`, `border-width-2 = 2px`. 라이트닝 등의 시스템에서는 주로 1px만 사용하여 `$border-width` 하나로 관리하거나, 굵은 focus용 2px 등 추가할 수 있습니다.
- **의미 토큰:** 맥락을 반영한 토큰. 예) `border.width.focus = 2px` (포커스시 두꺼운 테두리), `color.border.error = palette.red.500` (에러상태 테두리색) 등.
- **컴포넌트 토큰:** 컴포넌트별 테두리. 대부분 **색상이나 반경 토큰**으로 해결되므로 따로 두지 않아도 되나, 특정 컴포넌트의 테두리 두께가 전체와 다르면 토큰화할 수 있습니다 (예: `table.borderWidth = 2px` 등).

플랫폼 독립성: 테두리 토큰도 JSON으로 관리되어 CSS, Android, iOS에 반영됩니다. CSS에서는 `border-color`, `border-width` 등에 토큰을 매핑하고, iOS는 `layer.borderWidth`, Android는 `shape` 또는 `Paint` 등으로 적용합니다. 이때 단위 변환은 없으며, px는 dp/pt로 동일하게 여깁니다. 다만 **반투명도** 등 고려사항이 있다면 색상 토큰 단계에서 처리합니다. 테두리 스타일(`solid`, `dashed`)은 문자열 그대로 사용됩니다. JSON 예:

```
{
  "borders": {
    "width": {
      "base": { "value": 1 },
      "focus": { "value": 2 }
    },
    "color": {
      "default": { "value": "{palette.grey.500}" },
      "brand":   { "value": "{color.brand.primary}" }
    }
  }
}
```

위처럼 정의해두면 `border.width.base`는 1px, `border.width.focus`는 2px으로 쓰이고, 색상도 통합 관리됩니다. **플랫폼에 무관**하게 한 소스에서 정의되므로, 테두리 두께를 1px에서 2px로 올리는 디자인 변경도 한 줄 수정으로 가능합니다.

브랜드 확장성과 유지보수: Border 토큰에서는 주로 **색상이 브랜드 영향**을 받습니다. 브랜드 컬러가 바뀌면 Border에도 적용되는데, 예를 들어 `color.border.brand` 토큰을 사용하면 브랜드 팔레트에 맞춰 테두리색이 자동으로 바뀝니다 [55] [56]. Salesforce Lightning에서는 브랜드 색 관련 토큰이 여러 개 있는데, `BRAND_DISABLED` 등 토큰으로 **비활성 상태의 브랜드 테두리색**까지 지정하여 테마 변경 시 함께 변경되도록 했습니다 [57]. 또한 **포커스 표시**와 같이 접근성 관련 스타일도 토큰화해두면, 모든 컴포넌트에 일관된 굵기/색상으로 적용되고 한번에 조정할 수 있어 유지보수가 쉽습니다. 예컨대 "포커스 링 두께를 3px로 늘리자"고 하면 `border.width.focus` 토큰만 바꾸면 됩니다. 테마마다 포커스 색을 다르게 하는 것도 `color.border.focus` 토큰으로 제어할 수 있습니다.

토큰 명명 규칙과 예시: 테두리 토큰은 *border*라는 말을 포함하여 일관되게 짓습니다. 예) `border-width-1`, `borderWidthSmall` 등 두께, `color-border-primary`, `borderColor.error` 등 색상. Lightning에서는 색상 토큰에 `border` 접두를 붙여 `$color-border-input` (입력필드 테두리색)처럼 사용하고, 이 토큰들은 Global Access 여부와 함께 문서화되었습니다 [58] [59]. Uber Base Design System에서도 **회색 팔레트**를 background/content/border로 나눈 것처럼 [15], 색상 토큰 차원에서 border용 색들을 관리합니다. 일반적으로 **테두리 두께**는 값이 한두 개뿐이라 네이밍이 단순하지만, 시스템 전체 통일성을 위해 이 역시 토큰화합니다. 예시: Atlassian 디자인 시스템에서는 `border.width.0 = 0px (없음)`, `border.width.100 = 1px`, `border.width.200 = 2px` 식으로 네이밍해 숫자가 높아질수록 두꺼운 개념을 주었습니다. 이러한 규칙은 팀 내 디자인 철학에 맞게 정의하면 됩니다. 핵심은 **토큰을 통한 일관 관리**로, 한 컴포넌트라도 하드코딩된 테두리 값이 없도록 하는 것입니다.

Motion (모션)

구조와 계층: Motion 토큰은 **애니메이션 및 전환**에 쓰이는 시간(duration)과 가속 곡선(easing) 등을 정의합니다. 성숙한 디자인 시스템에서는 모션도 디자인 언어의 일부로 관리하여, **일정한 속도와 리듬**을 유지합니다. 일반적으로 **지속 시간(duration)**을 짧은 것부터 긴 것까지 서열화하고, **이징 곡선**도 표준 몇 개를 지정합니다. 예를 들어 Lightning의 Time 토큰을 보면 `$duration-instantly (0s)`, `$duration-immediately (0.05s)`, `$duration-quickly (0.1s)`, `$duration-promptly (0.2s)`, `$duration-slowly (0.4s)` 등 사람이 느끼는 감각에 따라 이름을 붙인 토큰들이 있습니다 [60] [61]. 이들은 각각 밀리초 값으로 정의되어 JSON에 저장되고, CSS나 JS 애니메이션, iOS Core Animation, Android Animators에서 사용됩니다. Easing의 경우 Material Design에서는 `standard`, `decelerate`, `accelerate` 세 종류 베지어 곡선을 권장하고, 이를 토큰 `easing.standard = cubic-bezier(x,x,x,x)` 형태로 관리할 수 있습니다.

- **원시 토큰:** 시간/곡선 등의 수치 값. 예) `duration.short = 100ms`, `duration.long = 400ms`, `easing.linear = cubic-bezier(0,0,1,1)`, `easing.standard = cubic-bezier(0.4, 0, 0.2, 1)` 등.
- **의미 토큰:** 사용 상황에 따른 토큰. 예) `motion.fadeIn.duration = duration.short`, `motion.drag.easing = easing.decelerate` 등으로 특정 상호작용 맥락의 애니메이션에 이름을 붙일 수 있습니다. 다만 많은 시스템에서는 굳이 맥락별로 쪼개지 않고, 공용 duration/easing 토큰만 정의하여 컴포넌트에서 조합하여 사용합니다.
- **컴포넌트 토큰:** 컴포넌트 전용 모션 값. 예) `tooltip.appear.duration = 150ms` 등. 일반 원칙은 **공용 토큰 재사용**이지만, 특수 사례에 한해 둘 수 있습니다.

플랫폼 독립성: 모션 토큰은 단위가 ms(밀리초) 또는 s(초)이므로 플랫폼 변환시에도 동일 단위를 사용합니다. JSON에 `duration.short = 0.1` (단위 s)로 저장하면, 웹 변환시 `0.1s`로, iOS에서는 0.1로 전달 후 코드에서 `UIView.animate(withDuration: token)` 식으로 씁니다. Easing은 문자열 bezier나 키워드("ease-in-out")를 그대로 토큰화하여 변환시 사용합니다. **프레임 수 기준**으로도 문서화할 수 있는데, Lightning 문서에서는 0.1초를 6프레임

(60fps 기준) 등으로 병기하여 이해를 돕습니다 [61]. 여러 플랫폼에서 동일한 타이밍을 느끼게 하려면 이러한 토큰을 일원화하는 것이 중요합니다.

브랜드 확장성과 유지보수: 브랜드별 모션 철학이 다를 수 있습니다. 어떤 브랜드는 역동적인 빠른 모션을, 어떤 곳은 침착한 느린 모션을 지향합니다. 토큰으로 관리하면 브랜드나 테마에 따라 **모션 토큰 세트**를 바꿀 수 있습니다. 예를 들어 어린이 대상 앱이라 모션을 천천히 보여주고 싶다면 `duration.short` 등을 2배 값으로 가진 별도 테마를 적용하면 됩니다. 또한 **사용자 선호** (motion reduction, 애니메이션 최소화 설정 등)에 대응하기 위해 모션 토큰을 central control point로 삼아, 환경에 따라 값 자체를 0으로 줄이는 것도 가능합니다. 유지보수 면에서는, "모든 hover 애니메이션을 조금 더 빠르게"와 같은 요청에 `duration.hover` 토큰값 한 번 수정으로 전체 반영이 이뤄집니다.

토큰 명명 규칙과 예시: 시간 토큰 명명은 **길이에 따른 수식어** 를 사용합니다. Lightning처럼 인간 친화적으로 (`immediately`, `quickly`, `slowly`) 할 수도 있고 [61] [62], 숫자로 단계화해서 `duration.level1/2/3`으로 할 수도 있습니다. Easing 토큰은 `easing.easeIn`, `easing.easeOut` 등 **동작 기술**을 이름에 넣습니다. Atlassian의 예를 들면 `motion.spring` (스프링 애니메이션), `motion.bounce` 등을 둘 수도 있겠죠. 중요한 것은 네이밍이 팀에서 **일관되고 이해하기 쉬울 것**입니다. Uber 등 일부 시스템에서는 모션 토큰도 **시스템(semantic) vs 플랫폼** 구분을 하는데, 예컨대 iOS 고유 모션이 있을 경우 토큰에 명시하거나 하기도 합니다. 그러나 대부분은 platform-agnostic하게 `motion.duration.xshort`, `motion.easing.standard`로 정의하고 각 플랫폼에 같은 개념으로 전달합니다.

예시로, **Salesforce Lightning**의 모션(duration) 토큰을 보면 다음과 같습니다:

- `$duration-immediately` – **거의 즉시**: 0.05s (3프레임) [60]
- `$duration-quickly` – **매우 짧게**: 0.1s (6프레임) [61]
- `$duration-slowly` – **천천히**: 0.4s (24프레임) [62]

이러한 토큰을 사용해 애니메이션 지속 시간을 통일하면, 개발자가 감으로 시간을 지정하는 일이 줄고, **디자인 의도대로 일괄 조정된 모션**을 구현할 수 있습니다.

State (상태에 따른 스타일)

구조와 계층: 컴포넌트의 **상태**(예: 기본, Hover, Active, Disabled 등)에 따른 스타일 변화를 디자인 토큰으로 관리하는 방식에는 **두 가지 접근**이 있습니다 [63] [64]:

1. **상태별 개별 토큰 정의 (Stateful Tokens)** – 각 상태 조합마다 별도의 토큰을 만드는 방법입니다. 예를 들어 `color.button.primary.default`, `color.button.primary.hover`, `color.button.primary.pressed`처럼 **토큰 이름에 상태** 를 포함하는 것입니다 [18]. Atlassian Atlas Design System이나 Adobe Spectrum 등이 이 방식을 쓰며, 토큰명만 봐도 해당 상태의 스타일을 바로 알 수 있다는 장점이 있습니다 [65]. 디자이너도 Figma 등에서 `bg-primary-hover` 같은 토큰을 보면 "Primary 버튼의 Hover 배경색"임을 즉각 이해할 수 있습니다 [66]. 하지만 상태와 테마가 늘어날수록 토큰 수가 폭증하는 단점이 있습니다. 예컨대 Light/Dark 모드 두 가지를 지원하려면 `*-hover` 토큰이 `*-hover-dark`까지 **2배로 증가**하고, 새로운 브랜드 추가 시 또 곱해집니다 [67] [68]. 그렇게 되면 관리가 어려워지고, Hover 상태 공통 변경 같은 요청에도 수십 개 토큰을 일일이 수정해야 할 수 있습니다 [68].

2. **상태 레이어 토큰 (Composable Tokens)** – 상태 자체를 하나의 **레이어 개념**으로 보고, **기본 스타일 + 상태 효과**를 조합하는 방법입니다 [69]. 이 접근에서는 `color.primary` 같은 기본색 토큰과 별도로

`state.hover.opacity = 0.08` 같이 **상태용 토큰**을 정의합니다 [70]. 예를 들어 Material Design 3에서는 **Pressed**, **Hover** 등의 state layer에 일정 투명도의 오버레이를 씌우는 기법을 사용하는데, 토큰으로 `state.hover = 8%` 투명도, `state.pressed = 12%` 투명도 등을 관리합니다 [71]. 그러면 Hover 상태의 색은 기본색 + 흰색 *8%* 오버레이로 표현되고, Dark 모드일 때는 반대로 기본색 + 검정 *8%* 오버레이로 처리하는 식으로 **토큰 조합**으로 구현합니다. 이 방식은 상태 이름을 토큰에 넣지 않으므로 토큰 종류가 훨씬 줄어들고, 다크모드나 새 테마 추가 시에도 **기본색 토큰 교체로 일괄 대응**할 수 있는 **고확장성** 장점이 있습니다 [2]. 실제로 Google의 Material Design 3는 이러한 **Layered 토큰** 접근을 택하고 있으며 [71], 새로운 브랜드나 테마가 추가되어도 상태 토큰(투명도 값 등)은 바꿀 필요 없이 기본 팔레트만 바꾸면 Hover/Pressed 효과가 자동으로 적용됩니다 [2].

많은 성숙한 디자인 시스템에서는 현실적으로 두 방식을 **혼합(Hybrid)**하기도 합니다 [72]. 핵심 UI 요소(예: Primary Button 등)는 가독성을 위해 `button.primary.hover`와 같이 **상태별 토큰**을 갖게 하고, 나머지 덜 핵심적인 부분은 **상태 레이어 토큰**을 조합하여 쓰도록 유연하게 운영합니다 [73] [74]. 이렇게 하면 토큰 수를 억제하면서도 필요한 곳엔 명시적 토큰을 제공하여 이해를 돕는 균형을 취할 수 있습니다.

플랫폼 독립성: 상태 토큰 자체는 플랫폼 구애를 받지 않습니다. JSON 등에 `state: { hover: { opacity: 0.08 } }`로 정의하고 [71], 웹에선 CSS에서 `:hover` 시에 `opacity: var(--state-hover)` 식으로 쓰거나, or `background-color: color.primary + opacity overlay`를 계산합니다. iOS/Android도 마찬가지로, 토큰 값을 가져와 상태별 스타일에 적용합니다. 물론 상태를 적용하는 방식(예: CSS :hover vs 모바일의 Touch feedback)이 다르지만, **토큰이 전달하는 값** (예: 밝게/어둡게 할 정도, 투명도 등)은 통일됩니다.

브랜드 확장성과 유지보수: 상태 스타일 역시 테마마다 달라질 수 있습니다. 예를 들어 기본 테마에선 Hover 시 살짝 어둡게, 다른 브랜드 테마에선 테두리만 강조하거나 하는 식의 차이가 있을 수 있습니다. 상태 토큰 접근 1 (상태별 토큰)에서는 이런 차이를 반영하려면 테마별 토큰을 각각 관리해야 합니다. 반면 접근 2 (레이어 토큰)에서는 상태 표현 로직이 **공통**이고 기본색만 달라지므로, 브랜드마다 **팔레트만 변경**하면 상태표현도 자연스럽게 어울리게 됩니다 [2]. 유지보수 측면에서도, 전체 시스템의 Hover 투명도를 8%→10%로 변경 같은 일이 생겼을 때 레이어 토큰 방식이면 `state.hover` 값 한 번 수정으로 끝나지만, 개별 토큰 방식이면 모든 `*-hover` 토큰을 찾아 수정해야 합니다 [68]. 따라서 규모가 크고 테마가 많은 시스템일수록 **레이어/Composable 토큰** 접근이 선호되는 추세입니다 [75].

토큰 명명 규칙과 예시: 상태 토큰을 개별 정의하는 경우, 토큰 이름에 **상태명을 suffix로 명시**합니다 (예: `color.primary.hover`, `border.success.disabled` 등) [76]. 이러한 네이밍은 직관적이지만, 앞서 말했듯 토큰 종류가 늘어날 수 있습니다. 반면 상태 레이어 토큰은 `state.hover = 0.08`처럼 독립된 카테고리로 관리합니다 [71]. Material Design 3에서는 토큰 이름보다는 **설계 가이드**로서 "hover시는 항상 내용 색상의 8% 투명 백색 오버레이"라고 정해 놓았습니다 [77]. 이를 토큰화하면 `state.hover.opacity = 0.08`, `state.focus.opacity = 0.12` 등으로 표현할 수 있을 것입니다. 실제 JSON 예시:

```
"state": {
  "hover":   { "opacity": 0.08 },
  "pressed": { "opacity": 0.12 },
  "disabled":{ "opacity": 0.38 }
}
```

위 JSON처럼 상태별 투명도를 정의해두고, 컴포넌트 스타일에서는 기본 배경+ `opacity: var(--state-hover)` 오버레이로 Hover 상태를 구현하는 식입니다 [71]. 이러한 구조를 활용하면 **토큰 조합으로 상태 표현**이 가능해져 매우 유연합

니다. 예를 들어 새 브랜드에서 기본 색이 바뀌어도 hover 오버레이 8%는 그대로 적용되므로, **일관된 상호작용 피드백**을 유지하면서도 색상만 브랜드에 맞게 변합니다 [2].

마지막으로, **상태 토큰 관리**의 모범 사례는 디자인 시스템 문서를 통해 팀에 공유되어야 합니다. 예를 들어 Atlassian은 대부분 토큰에 상태를 붙여 사용하면서도 다크 모드 토큰을 체계적으로 네이밍하여 (`.inverse` 또는 `-dark` 첨가) 관리 복잡도를 줄였습니다. 반면 Google Material 팀은 "상태는 조합으로 처리하고 토큰은 최소화"하는 전략을 취했습니다 [75]. 성숙한 시스템일수록 이러한 원칙을 명확히 정의해 개발자, 디자이너 모두가 **토큰 활용에 혼선을 갖지 않도록** 합니다.

참고 자료: 디자인 토큰의 개념과 구조에 대해서는 Contentful의 디자인 토큰 가이드 [3] [4], Uber Base Design System 사례 [15] [8], Salesforce Lightning Design System 토큰 문서 [59] [13] 등이 유용합니다. 또한 **상태 토큰**의 두 가지 접근법 비교는 Design Systems Collective 블로그의 "Stateful vs. Composable Tokens" 기사에 잘 나와 있으므로 참고하면 좋습니다 [78] [71]. 디자인 토큰을 체계적으로 적용하면, Netflix, Uber, Google, Salesforce와 같은 대규모 제품군에서도 **브랜드 아이덴티티의 일관성**을 유지하면서 **개별 팀의 생산성을 향상**시킬 수 있습니다.

---

[1] [3] [4] [6] [7] [11] Design tokens explained (and how to build a design token system) | Contentful
https://www.contentful.com/blog/design-token-system/

[2] [18] [19] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [78] The Story of Design Tokens: Stateful vs. Composable Tokens | by Amirhosein imeni | Design Systems Collective
https://www.designsystemscollective.com/the-story-of-design-tokens-stateful-vs-composable-tokens-4f8a3f736932?gi=31a131c2ec3d

[5] [12] The Pyramid Design Token Structure: The Best Way to Format, Organize, and Name Your Design Tokens | by Stefanie Fluin | Medium
https://stefaniefluin.medium.com/the-pyramid-design-token-structure-the-best-way-to-format-organize-and-name-your-design-tokens-ca81b9d8836d

[8] [15] [16] [17] [30] [53] [54] Cracking the Code of Color: Tokens, Systems, and Styles in Figma | by Kanvi Shaileshkumar Makwana | Bootcamp | Medium
https://medium.com/design-bootcamp/cracking-the-code-of-color-tokens-systems-and-styles-in-figma-38e9cc97eeab

[9] [10] Systematic Taxonomy in Design Tokens: A Framework for Scalable UI Architecture | by Gulshan Rahman | Design Systems Collective
https://www.designsystemscollective.com/systematic-taxonomy-in-design-tokens-a-framework-for-scalable-ui-architecture-45cc6f2c7686?gi=171994e04417

[13] [14] [21] [22] [23] [24] [25] [26] [27] [28] [29] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [55] [56] [57] [58] [59] [60] [61] [62] Design Tokens - Lightning Design System
https://design-system-site-summer-21.herokuapp.com/design-tokens/

[20] Base -Uber Design System - Gerardo Diaz
http://gerardodiaz.me/base

[77] States – Material Design 3
https://m3.material.io/foundations/interaction/states/state-layers