ChatGPT

Layout 체계를 Design Token처럼 구조화하기

Layout도 토큰화해야 하는 이유

현대 디자인 시스템에서는 **디자인 토큰**(Design Tokens)을 통해 색상, 타이포그래피, 간격 등을 중앙에서 관리합니다. 이처럼 **레이아웃**도 토큰처럼 체계화하면 디자인과 개발 간 소통이 원활해지고, 새로운 화면을 만들 때 **"어떤 레이아웃을 써야 하지?"**라는 고민을 줄여줍니다 [1]. 레이아웃 토큰은 화면 배치를 추상화하여 이름을 부여한 것으로, 팀이 공통의 **레이아웃 언어**를 갖도록 도와줍니다 [2]. 예를 들어 "대시보드형 레이아웃", "폼 입력용 집중 레이아웃"처럼 화면의 **용도에 따라 레이아웃 타입을 명명**하면, 어떤 용도에 어떤 레이아웃을 쓸지 쉽게 결정할 수 있습니다 [2]. 이를 통해 화면들이 목적에 맞는 구조를 갖게 되고, 여기저기서 제각각 만든 듯한 **일관성 없는 UI**를 예방할 수 있습니다 [3] [4].

또한 레이아웃을 토큰화하면, 코드 상에서 레이아웃 구조를 **JSON 등 데이터 형식으로 정의**하여 사용 가능해집니다. 디자인 변경 시에도 JSON 토큰 값만 수정하면 여러 화면에 반영되므로 유지보수가 수월해집니다 [5] [6]. 예를 들어 Salesforce는 **디자인을 한 곳에 JSON 토큰으로 정의**해 모든 플랫폼에 반영하는 Single Source of Truth를 구축했는데 [7] [8], 이처럼 레이아웃까지 토큰으로 관리하면 **디자인-개발 싱크 오차**를 크게 줄일 수 있습니다.

레이아웃 계층 구조: Shell에서 컴포넌트까지

레이아웃 토큰화를 이해하려면 UI 레이아웃의 **계층적 구조**를 떠올릴 필요가 있습니다. 일반적으로 다음과 같은 4단계 계층으로 구분할 수 있습니다:

- **Level 1 – Shell (앱 전체 프레임)**: 앱 공통으로 반복되는 최상위 구조를 의미합니다. 예를 들어 웹에서는 헤더+사이드바로 이루어진 **앱 쉘**이나 랜딩 페이지의 **마케팅 쉘** 등이 있고 [9], 모바일에선 탭 바 구조, 내비게이션 스택 구조 등이 해당됩니다. Carbon Design System의 "UI Shell"이 좋은 예로, 모든 제품이 공유하는 헤더/내비게이션 틀을 제공하여 앱 전역의 일관된 골격을 만듭니다 [9]. iOS도 **탭 바**, **내비게이션 스택**, **드로어** 등 표준 구조를 제시하여 앱들이 일관된 네비게이션 패턴을 따르도록 권장합니다. 이러한 Shell 토큰은 "전역 네비게이션 유형"처럼 정의될 수 있습니다.

- **Level 2 – Page Layout (페이지 레이아웃)**: 개별 화면(페이지)의 뼈대를 이루는 레이아웃입니다. 예컨대 **대시보드 화면 레이아웃**, **폼 입력 화면 레이아웃**, **리스트+디테일 화면 레이아웃** 등이 여기에 속합니다. 이는 화면을 몇 개 섹션으로 나눌지, 사이드바나 탭으로 구성할지 등을 정의합니다. 실제 사례로, Melio라는 SaaS 제품에서는 화면을 그 목적에 따라 **Job (작업 수행형)**, **Resource (목록/리소스 열람형)**, **Dashboard**, **Settings** 등 6가지 레이아웃 유형으로 분류해 두었고 [10] [11], Airbnb나 Asana도 각각 "여러 단계로 이루어진 작업(form wizard)"이나 "우측 하단 모달로 작업 생성" 같은 고유 패턴을 모든 유사 화면에 일관되게 적용했습니다 [12] [13]. **페이지 레이아웃 토큰**은 이러한 화면 유형들을 JSON에서 키값으로 정의해두고 (`"dashboard"`, `"form-wizard"` 등), 어떤 화면에 어떤 레이아웃을 쓸지 지정하는 식입니다.

- **Level 3 – Section/Layout Pattern (섹션 배치 패턴)**: 페이지 내부를 구성하는 섹션들의 배치 방식입니다. 예를 들어 **그리드(grid)** 2열, 3열, **스플릿(split)** 좌우 분할 30/70, **스택(stack)** 수직 스택 등 반복적으로 쓰는 섹션 배치 패턴들이 있습니다. 디자인 시스템에서는 이러한 패턴별 CSS 유틸리티나 컴포넌트를 제공하기도 합니다 (예: `grid-cols-3`, `flex flex-col center` 등). 이를 **섹션 레이아웃 토큰**으로 보면, `"grid-3"` = 3열 그

리드 + 표준 간격, `"split-30-70"` = 컬럼비 30/70 분할, `"stack-center"` = 중앙 정렬 스택 등으로 JSON에 정의해둘 수 있습니다 [14] [15]. 그러면 디자이너나 개발자가 "이 부분은 2열 그리드 섹션"이라고 토큰을 선택해 적용함으로써, 일관된 간격과 반응형 동작을 얻습니다.

- **Level 4 – Component Layout (컴포넌트 내부 레이아웃)**: 개별 컴포넌트 내부의 레이아웃입니다. 버튼 내부 아이콘+텍스트 정렬, 카드 컴포넌트 내부에서 이미지와 텍스트의 배치 등 **컴포넌트 단위** 레이아웃이 해당됩니다. 이는 주로 컴포넌트 구현 내부에서 Flexbox나 Inline-block으로 해결하며, 디자인 시스템 차원에서는 **컴포넌트별 권장 레이아웃** 가이드로 문서화됩니다. 예를 들어 "이미지와 텍스트가 있는 리스트 아이템은 좌측에 미디어, 우측에 텍스트를 두는 패턴"을 정의할 수 있습니다. 이 레벨에서는 개별 토큰이라기보다는 **컴포넌트의 layout props**나 가이드로 구현됩니다.

위 계층 각각을 토큰/패턴으로 정의하면, **Shell 토큰** 조합 + **Page 토큰** + **Section 패턴 토큰**만으로 화면의 골격을 기술할 수 있게 됩니다. 실제 구현에서 Next.js 같은 프레임워크는 **레이아웃 컴포넌트**를 중첩하여 이러한 계층을 구성하며, Design System 문맥에서는 JSON 스키마로 각 화면의 구성요소를 선언하는 접근도 가능합니다.

디자인 시스템의 레이아웃 토큰 활용 사례

여러 **성숙한 디자인 시스템**에서 레이아웃을 체계화한 사례를 찾아볼 수 있습니다. 대표적으로:

- **Carbon Design System (IBM)** – Carbon은 8px 기반의 **2× Grid 시스템**과 표준 **브레이크포인트**를 정의하고 있습니다. Small, Medium, Large 등 다섯 단계의 반응형 breakpoint를 명시하여 각 해상도에서 컬럼 수, 마진, 패딩 규칙이 일관되게 적용됩니다 [16]. 이처럼 **브레이크포인트** 값 자체를 토큰으로 관리하면 (`small: 320px`, `medium: 672px` 등) 레이아웃 전환 기준을 코드와 디자인에서 동일하게 사용할 수 있습니다. 또한 Carbon은 **UI Shell** 컴포넌트를 제공하여, 모든 애플리케이션이 공통으로 쓰는 헤더/내비게이션 레이아웃 틀을 쉽게 구축하도록 했습니다 [9]. 이를 통해 제품들 간에 일관된 Shell 구조를 갖추면서도 필요에 따라 Left/Right 패널을 붙이거나 떼는 유연성을 제공합니다 [17].

- **Material Design 3 (Google)** – Material Design은 레이아웃을 위해 **Responsive Layout Grid**와 권장 **밀도 (Density)**를 제시하고, 최신 Material 3에서는 **Window Size Class** 개념을 도입했습니다. Compact, Medium, Expanded 세 가지 화면 크기 분류에 따라 1-3개의 **Pane**으로 UI를 나누는 패턴을 안내하고 있습니다 (예: 큰 화면에서는 리스트+디테일 2페인 레이아웃 적용 등) [18]. 또한 Material는 컴포넌트마다 적절한 breakpoints에서 **navigation 변형**(모바일 Bottom Nav → 태블릿 Nav Rail → 데스크탑 Permanent Nav 등)이 이루어지도록 규칙을 문서화하고 있습니다. 이처럼 OS나 프레임워크 차원 표준이 디자인 시스템에 녹아들어 **상황별 최적 레이아웃 패턴**을 토큰화한 사례라고 볼 수 있습니다.

- **Fluent UI (Microsoft)** – Fluent Design System에서는 **4px 공통 간격 단위**를 모든 컴포넌트와 레이아웃에 적용하여 일관성을 유지합니다 [19]. Fluent의 전역 **Spacing ramp**는 4px 베이스로 짜여진 일련의 간격 토큰 (`size40=4px`, `size160=16px`, `size400=40px` 등)을 제공하며, 컴포넌트 내부, 패턴 간, 레이아웃 사이 각 용도에 맞는 권장 spacer를 나눠두고 있습니다 [20] [21]. 예를 들어 작은 컴포넌트 내부 요소 사이에는 `size40(4px)`처럼 좁은 간격을, 화면 섹션 사이에는 `size160(16px)` 이상의 넉넉한 간격을 주어 시각적 그룹화를 합니다 [22] [23]. 또한 **Grid** 시스템도 12컬럼을 기본으로 컬럼/거터/마진을 정의하여, 다양한 화면에서 **일관된 비례**를 유지합니다. 이러한 값들은 모두 Fluent의 **디자인 토큰**으로 정의되어 여러 플랫폼(iOS pt, Android dp 등)에서 동일한 의미로 쓰입니다 [24] [25].

- **기타 예시**: Adobe Spectrum 디자인 시스템도 **분할뷰, 모달, 탐색 패턴** 등을 가이드하며, IBM Carbon이나 Ant Design처럼 **페이지 레이아웃 템플릿**을 제공하기도 합니다 (예: Ant Design ProLayout 컴포넌트). GitLab의 Pajamas 디자인 시스템에서는 **Application Chrome (글로벌 네비게이션 영역)**과 **패널 구성** 등을 명시적으로 설계하여 복잡한 인터페이스도 일정한 규격 안에서 구성하도록 합니다 [26] [27]. 이렇듯 성숙한 디자인 시스템들은 레이아웃에서 흔히 발생하는 패턴들을 미리 정의하고, **디자인 가이드 + 코드 컴포넌트** 형태로 지원하여 **일관성**과 **재사용성**을 높이고 있습니다.

특히 **간격(Spacing)**과 **그리드**는 대부분의 디자인 시스템에서 토큰화된 중요한 레이아웃 요소입니다. 예를 들어 Sainsbury's 디자인 시스템은 8px 기반 **Baseline Grid**와 **Size/Spacing tokens**를 제공하여 모든 마진/패딩이 정의된 값만 쓰이도록 규정하고 있습니다. 앞서 소개한 PLUS 사례에서는 컴포넌트 내부, 컴포넌트 사이, 섹션 사이 등 **맥락별 간격 토큰**을 세분화하여 혼용을 막고 [28] [29], 2px 혹은 4px 단위 증분으로 엄격히 제한한 spacing scale을 도입했습니다 [30] [31]. 그 결과 디자이너들이 의미에 맞는 간격을 고르게 되고, 작은 컴포넌트에 과도한 큰 간격을 넣는 **"의미적 불일치"**를 줄였다고 합니다 [32] [33].

또 다른 예로, **미국 정부의 USWDS**도 타이포 스케일, spacing scale, 그리고 레이아웃 그리드를 토큰으로 지정하여 공공 웹사이트들이 공통의 레이아웃 접근을 따르도록 돕습니다. 이처럼 **레이아웃 토큰**을 활용하면 팀원들 (디자이너, 개발자 모두)이 "어떤 간격, 어떤 그리드, 어떤 화면틀을 써야 하지?"를 쉽게 답할 수 있는 **공통 언어와 도구**를 얻게 됩니다 [4] [34].

JSON을 활용한 레이아웃 토큰 구현

이제 레이아웃 토큰을 **JSON 등 구조화된 데이터**로 관리하는 방안을 살펴보겠습니다. W3C의 Design Tokens 표준이나 Style Dictionary 같은 도구를 사용하면, 디자인 토큰을 플랫폼에 중립적인 JSON/YAML로 정의하고 이를 다양한 플랫폼의 코드 변수로 변환할 수 있습니다 [7] [35]. 일반적으로 디자인 토큰 JSON에는 색상, 폰트와 함께 **spacing, breakpoint, grid** 정보도 포함됩니다 [5] [6]. 즉, 레이아웃 관련 토큰들도 **이름-값 쌍**으로 JSON에 저장해둘 수 있다는 것입니다.

예를 들어 **Chakra UI v3**의 테마 설정(JSON)에서는 `tokens.grid` 섹션에 **반응형 레이아웃 토큰**을 정의합니다. 아래 예시는 JSON 토큰의 일부로, 그리드의 거터(gutter) 간격과 컬럼 수를 베이스(base)와 large(lg) 화면 크기별로 지정한 것입니다 [15]:

```
"grid": {
  "gutterSize": { "value": { "base": "1rem", "lg": "2rem" } },
  "count":      { "value": { "base": 4,       "lg": 12    } }
}
```

이처럼 **JSON 토큰**으로 `grid.gutterSize`나 `grid.count`를 정의해 두면, 개발자는 코드에서 이 값을 참조하여 작은 화면에서는 4컬럼 그리드에 1rem 간격, 큰 화면에서는 12컬럼에 2rem 간격을 자동으로 적용할 수 있습니다 [15]. **브레이크포인트**도 마찬가지로 JSON에 `breakpoints` 객체로 `{ sm: "640px", md: "768px", ... }` 식으로 넣어 두고, 미디어쿼리나 컨테이너쿼리에서 해당 토큰을 참조하도록 할 수 있습니다. 예컨대 Carbon의 디자인 토큰에도 작은/중간/큰 화면 폭에 대한 픽셀 값들이 정의되어 있는데 [16], JSON으로 관리하면 필요시 일괄 조정도 쉽고 여러 코드베이스에서 동일한 기준을 공유할 수 있습니다.

Tekton Layout Token 예시(`interface LayoutSystem { ... }`)처럼, 레이아웃 토큰 JSON 구조를 설계할 때는 계층별로 키를 나누어 정의하면 관리하기 좋습니다. 예를 들어:

- `shells.web.app`, `shells.web.marketing`, `shells.mobile.tab-bar` 등 **Shell 패턴**별 설정,
- `pages.dashboard.sections` 배열과 `pages.dashboard.responsive`처럼 **페이지 템플릿**별 섹션 구성과 반응형 설정,
- `sections["grid-3"] = "grid grid-cols-3 gap-{spacing}"`처럼 **섹션 레이아웃 패턴**별 CSS 유틸리티 값,
- `breakpoints.lg = "1280px"` 등의 **반응형 임계값**,

으로 JSON 스키마를 계층적으로 만들 수 있습니다. 이렇게 하면 하나의 **테마 번들**에 레이아웃 토큰 JSON이 포함되어, 디자인 시스템에서 테마를 교체하듯 레이아웃 전략도 통째로 교체할 수 있습니다. 실제로 한 디자인 시스템 사례에서는 "같은 테마 + 같은 페이지 타입" 선택 시 **웹과 모바일에 최적화된 레이아웃이 자동 적용**되도록 설계했는데, 이러한 **템플릿 번들** 개념이 가능하려면 레이아웃 토큰이 뒷받침되어야 합니다 (웹 대시보드는 2컬럼 그리드, 모바일 앱에서는 1컬럼 스택으로 자동 전환 등). JSON 토큰으로 각 상황별 레이아웃 구성을 미리 정의해두면 이런 **자동 Responsive 매핑**도 구현할 수 있습니다 [15].

정리하면, 레이아웃을 디자인 토큰처럼 관리하기 위해서는 **①레이아웃 요소의 표준화**, **②계층별 명확한 명명**, **③JSON 스키마화 및 코드 연동**이 필요합니다. 먼저 팀과 제품에 맞는 **레이아웃 분류체계**를 수립하고 [36], 각 분류 (shell/page/section 등)에 일관된 이름과 속성을 정의합니다. 그리고 그것을 JSON 토큰으로 작성하여, 디자이너는 Figma 등의 토큰 툴로 관리하고 개발자는 Style Dictionary나 전처리 스크립트로 해당 JSON을 가져와서 **코드 상수**로 변환해 사용하는 흐름을 만듭니다 [35] [37]. 이렇게 하면 디자인 변경 시 토큰 JSON만 업데이트하면 되고, **코드 에이전트**나 빌드 스크립트가 이를 파싱해 화면을 구성하므로 사람 손으로 코딩할 때 생길 수 있는 실수를 방지할 수 있습니다.

결론: 레이아웃 시스템의 구축과 문서화

레이아웃을 토큰화하고 체계화하는 것은 한 번에 완벽하게 이루어지지 않을 수도 있습니다. 우선 제품의 **기존 화면들을 모두 인벤토리**하면서 어떤 레이아웃 패턴들이 쓰이고 있는지 감사(Audit)하는 것이 출발점입니다 [38]. 그 다음 디자이너와 개발자가 모여 패턴들을 **추상화된 몇 가지 타입**으로 묶고, 명칭을 정하며, 왜 그런 레이아웃이 필요한지 **의도를 문서화**해야 합니다 [39] [40]. 필요한 경우 모달, 드로어같이 **보조 패턴**도 정의에 포함해야겠지요 [41]. 마지막으로 이러한 내용을 디자인 시스템 문서나 스토리북 등에 **시각적 예시와 함께 정리**합니다 [42].

잘 정비된 레이아웃 시스템을 갖추면 디자이너들은 새로운 화면을 만들 때 **재사용 가능한 레이아웃 템플릿**에서 시작할 수 있고, 개발자들은 JSON 토큰과 컴포넌트 조합만으로 신속하게 해당 레이아웃을 구현할 수 있습니다. 실제 현업 보고에 따르면, 레이아웃 토큰과 패턴을 도입한 후 **디자인 작업 속도가 향상**되고, 화면들이 구성요소들로 **빠르게 조립**되었으며 [43] [44], 새로운 팀원이 들어와도 토큰 이름만 보면 **어떤 레이아웃과 간격을 써야 할지 이해**하기 쉬워졌다고 합니다 [32] [31].

결과적으로 **레이아웃 토큰화**는 디자인 시스템을 한층 성숙하게 만드는 **마지막 퍼즐 조각**이라고 볼 수 있습니다. 색상이나 타이포처럼 눈에 보이는 스타일뿐 아니라, 보이지 않는 **공간 구조와 패턴**까지 시스템으로 관리하면 제품 전반의 UX 일관성이 높아집니다 [45]. 이는 디자이너에게는 명확한 가이드가 되고, 개발자에게는 신뢰할 수 있는 청사진이 되어 결국 사용자에게 **일관되고 직관적인 경험**으로 돌아오게 됩니다.

Sources: 디자인 시스템 토큰화 사례 및 레이아웃 시스템 참고 [1] [2] [46] [6] [16] [15]

---

1 2 3 4 10 11 12 13 34 36 38 39 40 41 42 45 Layout systems: The missing ingredient to your design system | by Vitsky | Melio's R&D blog | Medium
https://medium.com/meliopayments/layout-system-the-missing-ingredient-to-your-design-system-5a53c5d84653

5 6 7 8 35 Design Tokens – What Are They and How to Use Them? - Bejamas
https://bejamas.com/blog/design-tokens-what-are-they-and-how-to-use-them

9 17 Open menu
https://carbondesignsystem.com/components/UI-shell-header/usage/

14 15 37 Generating a Custom Chakra UI v3 Theme from Design Tokens: A Complete Guide - DEV Community
https://dev.to/kiranmantha/generating-a-custom-chakra-ui-v3-theme-from-design-tokens-a-complete-guide-1085

16 carbondesignsystem.com
https://carbondesignsystem.com/elements/2x-grid/overview/

18 Applying layout – Material Design 3
https://m3.material.io/foundations/layout/applying-layout

19 20 21 22 23 24 25 Layout - Fluent 2 Design System
https://fluent2.microsoft.design/layout

26 27 Layout | Pajamas Design System
https://design.gitlab.com/product-foundations/layout/

28 29 30 31 32 33 PLUS Design Tokens
https://yiyang.works/plus

43 44 46 Architecting clarity: a scalable design system for quick commerce | by Rahim | Muzli - Design Inspiration
https://medium.muz.li/architecting-clarity-a-scalable-design-system-for-quick-commerce-f145d0aa711d?gi=d005cfd87267