AI 기반 UI/UX 우수성: 깔끔하고 유용한 Dash 애플리케이션을 위한 Dash Mantine Components 가이드

1. 서론: Mantine Components를 통해 우수한 Dash UI/UX를 위한 AI 안내

이 문서는 Dash Mantine Components (DMC) 라이브러리 (dash==3.0.4 버전 사용)를 활용하여 깔끔하고 유용하며 미학적으로 만족스러운 Dash 애플리케이션을 개발하는 데 필요한 포괄적인 가이드 역할을 합니다. 핵심 목표는 "The UI_UX Playbook"에 상세히 설명된 확립된 UI/UX 모범 사례를 AI 기반 Dash 개발을 위한 실행 가능한 지침으로 전환하는 것입니다.

사용자 인터페이스 (UI)는 전체 사용자 경험 (UX)의 초기 분위기를 설정하므로 가장 중요합니다. 고품질 UI/UX 달성은 우연이 아니라 기본적인 디자인 원칙을 세심하게 적용한 결과입니다. AI의 경우, 이러한 원칙들이 특정 DMC 컴포넌트 사용 및 테마 전략에 매핑될 때, 우수한 인터페이스를 생성하기 위한 기본적인 "규칙"을 형성합니다. 미학적으로 놀라운 UI도 복잡한 탐색과 같은 좋지 않은 UX로 인해 비효율적이 될 수 있음을 인식하는 것이 중요합니다. 반대로, 뛰어난 UX가 평범한 UI와 결합되면 애플리케이션이 구식이고 매력적이지 않게 느껴질 수 있습니다. UI와 UX 모두 능숙하게 실행될 때 최적의 사용자 경험이 나타납니다. 이 가이드는 AI가 이러한 시너지를 달성하는 데 필요한 지식을 제공하는 것을 목표로 합니다.

AI를 대상으로 한다는 명시적인 지침은 "선언적인" 안내 스타일을 필요로 합니다. 이는 문서가 좋은 UI가 *무엇인지* 설명할 뿐만 아니라 특정 DMC 코드 패턴 및 속성 구성을 통해 이를 *어떻게* 달성하는지 세부적으로 설명할 것임을 의미합니다. 이러한 접근 방식은 AI에게 명확하고 모호하지 않은 지침을 제공하며, 이는 코드 생성에 중요합니다. 사용자가 "깔끔하고 유용한 Dash 애플리케이션을 만드는 데 AI를 돕는 유용한 프로젝트 지식으로 사용될 수 있는 .md 파일"을 요청한 것은 추상적인 원칙에서 구체적인 DMC 구현으로의 실용적인 변환 계층의 필요성을 강조합니다. 예를 들어, "시각적 계층 구조" 원칙은 "dmc.Title을 `order` 속성과 함께 사용하고, dmc.Text를 적절한 `size` 및 `weight` 속성과 함께 사용하여 `theme.headings` 객체가 일관된 제목 스타일을 위해 구성되도록 합니다"와 같은 지침으로 변환됩니다.

Dash Mantine Components는 현대 UI/UX 패러다임과 완벽하게 일치하는 풍부한 컴포넌트 세트와 강력한 테마 시스템을 제공합니다. dash==3.0.4 버전과 문서 작성 시점의 DMC 버전에 중점을 두는 것은 AI의 운영 지식이 이 특정 기술 컨텍스트 내에서 사용 가능한 기능 및 속성에 기반해야 함을 의미합니다. Dash 또는 DMC의 향후 업데이트는 유용성을 유지하기 위해 이 지침의 수정이 필요할 수 있습니다.

이 문서는 테마, 레이아웃, 데이터 표시, 타이포그래피, 데이터 시각화와 같은 주요 UI/UX 영역을 체계적으로 탐색할 것입니다. 각 섹션은 "The UI_UX Playbook"의 원칙을 참조하고 이를 실제 DMC 컴포넌트, 속성 및 테마 구성에 매핑하여 AI를 위한 구조화되고 실행 가능한 지식 기반을 제공합니다.

2. Dash Mantine Components를 이용한 핵심 테마

테마는 시각적으로 일관되고 매력적인 애플리케이션의 초석입니다. Dash Mantine Components (DMC)는 주로 `dmc.MantineProvider` 컴포넌트를 통해 관리되는 강력하고 유연한 테마 시스템을 제공합니다. 잘 정의된 테마는 AI가 미학적으로 만족스러울 뿐만 아니라 가독성 및 시각적 계층 구조와 같은 UX 원칙을 최소한의 컴포넌트별 구성으로 준수하는 UI를 생성하는 데 중요합니다.

DMC 테마의 핵심은 `dmc.MantineProvider`입니다. 모든 DMC 애플리케이션은 이 컴포넌트의 단일 인스턴스로 둘러싸여야 합니다. `dmc.MantineProvider`는 애플리케이션 전체에 테마를 전파하고, 색 구성표 (밝은/어두운 모드)를 관리하며, 전역 스타일 및 CSS 변수를 주입하는 역할을 합니다.

Key props for dmc.MantineProvider include:

- `theme` (dict): 모든 디자인 토큰 및 사용자 정의가 정의되는 중앙 테마 객체입니다. Mantine의 기본 테마와 깊게 병합되어 선택적 재정의가 가능합니다.
- `forceColorScheme` (string: "light" or "dark"): 이 속성을 통해 AI는 색 구성표를 프로그래밍 방식으로 설정할 수 있으며, 이는 일관된 미리보기를 생성하거나 특정 디스플레이 요구 사항이 있는 애플리케이션에 유용하며, 사용자 또는 시스템 기본 설정을 재정의합니다.
- `withGlobalStyles` (boolean): `True`로 설정하면 (권장) Mantine의 기본 전역 스타일을 적용하여 일관된 모양과 느낌을 제공합니다.
- `withNormalizeCSS` (boolean): `True`로 설정하면 (권장) 브라우저 불일치를 줄이기 위해 CSS 정규화 스타일 시트가 포함됩니다.
- `inherit` (boolean): `True`인 경우, `MantineProvider`는 DOM 트리의 부모 요소에서 글꼴 및 색상 스타일을 상속하려고 시도합니다.

2.1. 색상 시스템 정의

색상은 UI 디자인의 근본적인 측면으로, 분위기, 브랜드 인식 및 유용성에 영향을 미칩니다. 효과적인 색상 시스템은 명확성, 대비 및 일관성을 우선시합니다.

색상 사용 원칙:

- **절제:** "적을수록 좋다"는 핵심 원칙입니다. 기본 색상은 인터페이스를 압도하는 대신 버튼 및 링크와 같은 대화형 요소를 강조하기 위해 신중하게 사용해야 합니다.
- **배경:** 중립적인 배경색 (회색, 베이지색 또는 부드러운 흰색 음영)은 종종 최상의 캔버스를 제공하여 콘텐츠와 기본 색상이 돋보이게 합니다. 어두운 배경은 특히 저조도 조건에서 눈의 피로를 줄이고 텍스트를 돋보이게 하는 데 매우 효과적일 수 있습니다.
- **대비:** 텍스트와 배경, 그리고 다른 UI 요소 간의 충분한 대비는 가독성, 사용자 주의 유도, 시각 장애가 있는 사용자를 위한 접근성에 중요합니다.
- **의미론적 색상:** 시스템 상태에 대한 확립된 색상 규칙을 준수합니다: 오류는 빨간색, 성공은 녹색, 경고는 노란색을 사용하여 사용자에게 즉각적이고 직관적인 피드백을 제공합니다.

색상 시스템을 위한 DMC 구현:

- **`theme.primaryColor`** (string): 애플리케이션의 주요 강조 색상을 설정하는 테마 속성입니다. 값은 `theme.colors` 내에 정의된 키여야 합니다 (예: "blue" 또는 사용자 정의 "myBrandGreen"). 이 색상은 특정 색상이 제공되지 않을 때 `dmc.Button`과 같은 많은 대화형 컴포넌트에서 기본적으로 사용됩니다.
  - **AI 지침:** AI는 애플리케이션의 브랜드 아이덴티티를 기반으로 `primaryColor`를 정의해야 합니다. 이 색상은 사용자 집중을 유도하기 위해 주요 Call-to-Action 버튼, 활성 탐색 요소 및 기타 주요 대화형 컴포넌트에 일관되게 적용되어야 합니다.
- **`theme.colors`** (dict): 사용자 정의 색상 팔레트가 정의되거나 기존 Mantine 기본 색상 ("blue", "red", "green" 등)이 재정의되는 사전입니다. `theme.colors`의 각 항목은 0 (가장 밝은)에서 9 (가장 어두운)까지 색상 이름 (문자열 키)과 10개의 문자열 음영 배열이 연결되어야 합니다. 모든 10개의 음영을 제공하는 것은 컴포넌트가 다양한 상태 (호버, 활성)에서 변형 (예: light, filled, outline)을 올바르게 렌더링하는 데 필수적입니다.
  - 예시: `theme={"colors": {"myBrandBlue": ["#E7F5FF", ..., "#1864AB"], "deepPurple": [...]}}`
  - **AI 지침:** Mantine의 기본값에 없는 브랜드 색상의 경우, AI는 `theme.colors` 내에 정의해야 합니다. 각 사용자 정의 색상 배열에 정확히 10개의 음영이 포함되어야 합니다. 기본 헥스 색상만 사용 가능한 경우, Mantine 색상 생성기 (DMC 문서에 언급됨)와 같은 도구를 사용하여 10개의 음영을 생성할 수 있습니다.
- **`theme.primaryShade`** (number 또는 dict): `theme.primaryColor`의 어떤 음영 (인덱스 0-9)이 기본적으로 사용되는지 지정합니다. 단일 숫자 (예: `6`)로 밝은 및 어두운 색 구성표 모두에 적용하거나, 다른 밝은 및 어

두운 모드에 대해 다른 음영을 지정하는 사전 (예: `{light: 6, dark: 8}`)일 수 있습니다.
    - **AI 지침:** AI는 `primaryShade`를 조정하여 기본 색상이 시각적으로 보기 좋고, 중요하게, 밝은 및 어두운 테마 모두에서 기본 색상 배경의 텍스트 및 아이콘에 충분한 대비를 제공하도록 해야 합니다. 이는 접근성에 필수적입니다.
- **컴포넌트에 색상 적용:**
    - 개별 컴포넌트는 `color` 속성을 사용하여 색상을 설정할 수 있습니다. 이 속성은 테마 색상 이름 (예: `color="myBrandBlue"`), 특정 음영 인덱스가 있는 테마 색상 이름 (예: 더 어두운 빨간색을 위한 `color="red.7"`), 또는 직접 CSS 색상 값 (예: `color="#FF5733"`)을 허용합니다.
    - **AI 지침:** 기본 작업의 경우, AI는 `color=theme.primaryColor` (또는 `primaryColor`가 의도된 경우 `dmc.Button`과 같은 컴포넌트에서 `color` 속성을 생략)를 사용해야 합니다. 의미론적 색상 지정 (오류, 경고, 성공 메시지)의 경우, AI는 개요에 설명된 색상 규칙을 준수하여 `color="red"`, `color="yellow"`, `color="green"`과 같은 적절한 테마 색상을 사용해야 합니다.
- **색상에 대한 Mantine CSS 변수:** 사용자 정의 색상을 포함한 모든 테마 색상은 CSS 변수로 자동 노출됩니다. 이러한 변수는 `var(--mantine-color-{colorName}-{shadeIndex})` (예: `var(--mantine-color-myBrandBlue-5)`) 또는 기본 색상의 변형을 위한 `var(--mantine-primary-color-filled)`와 같은 패턴을 따릅니다.
    - **AI 지침:** 표준 컴포넌트 속성을 넘어서는 사용자 정의 CSS 스타일링 요구 사항이 있거나 테마와 일치하도록 비 Mantine 요소를 스타일링할 때, AI는 색상 일관성을 유지하기 위해 이러한 CSS 변수를 활용해야 합니다.
- **기본 및 대비 색상:**
    - `theme.black` 및 `theme.white`: 테마의 절대적인 검정색 및 흰색을 정의합니다.
    - `theme.autoContrast` (boolean) 및 `theme.luminanceThreshold` (number): `autoContrast`가 `True`이면, Mantine은 `luminanceThreshold`를 기반으로 색상 배경 (채워진 버튼 등)의 텍스트 색상 (검정색 또는 흰색)을 자동으로 조정하여 충분한 대비를 보장하려고 시도합니다.
    - **AI 지침:** AI는 대화형 요소에 다양한 배경색을 사용하는 애플리케이션의 경우 `autoContrast`를 활성화하는 것을 고려해야 합니다. 이는 가독성 및 접근성을 유지하는 데 도움이 되며, 대비 원칙과 일치합니다.

`theme.primaryColor`, `theme.colors`, `theme.primaryShade`의 신중한 정의 및 적용은 기본입니다. 이를 통해 AI는 브랜드 정렬을 달성하는 동시에 특히 밝은 및 어두운 모드를 고려할 때 적절한 색상 대비를 통해 접근성을 보장할 수 있습니다. AI는 `theme.colors`에 포괄적인 브랜드 색상 팔레트 (색상당 10개의 음영 포함)를 정의하는 것이 첫 번째 단계임을 이해해야 합니다. 그런 다음, 이들 중에서 `primaryColor`를 선택하고 다른 모드에 맞게 `primaryShade`를 미세 조정하면 기본 대화형 요소가 브랜드에 충실하고 사용 가능하도록 보장됩니다.

2.2. 타이포그래피 시스템

타이포그래피는 UI 디자인의 중요한 요소이며, 가독성, 사용자 참여 및 애플리케이션의 전반적인 미적 매력에 크게 영향을 미칩니다.

타이포그래피 원칙:

- **글꼴 선택:** 글꼴 선택은 애플리케이션의 컨텍스트 (예: 공식, 기술 중심, 창의적)를 고려하여 신중해야 합니다. 가독성과 시각적 매력을 우선시하십시오. 일관성을 유지하고 혼란스러운 모양을 피하려면 글꼴 선택을 한두 개로 제한하는 것이 일반적으로 가장 좋습니다.
- **줄 길이:** 최적의 줄 길이는 가독성을 향상시킵니다. 데스크톱 인터페이스의 경우 줄당 45-75자를 목표로 하고, 모바일의 경우 30-40자가 더 적합합니다.
- **텍스트 대비:** 흰색 배경에 순수 검정색 텍스트 (`#000000`) 또는 그 반대를 사용하지 마십시오. 눈의 피로를 줄이고 시각적 조화를 개선하려면 제목에는 오프 블랙 또는 진한 회색을, 본문 텍스트에는 약간 더 밝은 회색을 선택하십시오.

- **글꼴 두께**: 계층 구조를 설정하고 주의를 끌기 위해 제목에는 더 무거운 글꼴 두께 (예: 굵게, 세미 볼드)를 사용하십시오. 본문 텍스트에는 제목과 경쟁하지 않으면서 가독성을 보장하기 위해 더 가벼운 두께 (예: 일반)가 적합합니다.
- **크기 및 계층 구조**: 글꼴 크기는 중요도를 전달하는 주요 도구입니다. 본문 텍스트의 일반적인 표준은 16px입니다. 제목은 비례적으로 더 커야 합니다.
- **줄 높이 (Leading)**: 충분한 줄 높이는 가독성에 필수적입니다. 본문 텍스트 (약 16px)의 경우, 글꼴 크기의 1.5 ~ 1.6배 (150%~160%)의 줄 높이가 권장됩니다. 제목은 종종 약간 더 좁은 줄 높이를 사용할 수 있지만, 단락은 더 넉넉한 간격에서 이점을 얻습니다.
- **텍스트 정렬**: 긴 텍스트의 경우, 왼쪽에서 오른쪽으로 읽는 언어에서는 각 줄의 일관된 시작점을 제공하여 가독성을 돕기 때문에 일반적으로 왼쪽 정렬이 선호됩니다.

타이포그래피를 위한 DMC 구현:

- `theme.fontFamily` (string): 애플리케이션의 대부분의 컴포넌트에 대한 전역 글꼴 모음을 정의합니다. 광범위한 호환성 및 성능을 위해 시스템 글꼴 스택 또는 'Inter'와 같은 특정 웹 글꼴을 사용하는 것이 일반적인 관행입니다.
  - 예시: `theme={"fontFamily": "Inter, sans-serif"}`
  - **AI 지침**: AI는 `fontFamily`를 가독성이 높은 산세리프 글꼴로 설정해야 합니다. 시스템 글꼴 (예: `-apple-system, BlinkMacSystemFont, Segoe UI, Roboto,...`)을 사용하는 것이 성능과 네이티브 느낌을 위한 좋은 기본값입니다.
- `theme.fontFamilyMonospace` (string): `dmc.Code`, `dmc.Kbd`, `dmc.CodeHighlight`와 같이 코드를 표시하는 컴포넌트의 글꼴 모음을 지정합니다.
  - **AI 지침**: AI는 명확하고 가독성이 좋은 모노스페이스 글꼴 스택 (예: `ui-monospace, SFMono-Regular, Menlo,...`)을 선택해야 합니다.
- `theme.headings` (dict): 이 객체는 `dmc.Title` 컴포넌트 (`<h1>`-`<h6>` HTML 태그로 렌더링됨)의 전역 스타일링을 허용합니다. 다음을 포함할 수 있습니다.
  - `fontFamily` (string): 원하는 경우 `theme.fontFamily`를 재정의하여 모든 제목에 대한 특정 글꼴 모음.
  - `fontWeight` (string | number): 모든 제목의 기본 글꼴 두께.
  - `sizes` (dict): `h1`에서 `h6`까지 각 제목 수준에 대한 특정 `fontSize` 및 `lineHeight` (선택적으로 `fontWeight`)를 정의하는 중첩 사전.
    - 예시: `theme={"headings": {"fontFamily": "Georgia, serif", "fontWeight": 700, "sizes": {"h1": {"fontSize": "2.5rem", "lineHeight": 1.3}, "h2": {"fontSize": "2rem", "lineHeight": 1.35, "fontWeight": 600},...}}}`
  - **AI 지침**: AI는 디자인 원칙에서 강조된 명확한 시각적 계층 구조를 설정하기 위해 `theme.headings.sizes`를 구성해야 합니다. 글꼴 크기는 `h1`에서 `h6`까지 점진적으로 감소해야 합니다. 줄 높이는 일반적으로 더 큰 글꼴 크기에 대해 더 좁아야 합니다. 글꼴 두께도 추가적인 구분을 위해 제목 수준별로 변경할 수 있습니다.
- `theme.fontSizes` (dict): `dmc.Text` 및 기타 컴포넌트에서 `size` 속성을 통해 사용할 수 있는 미리 정의된 글꼴 크기 토큰 (일반적으로 'xs', 'sm', 'md', 'lg', 'xl')의 스케일을 정의합니다.
  - 예시: `theme={"fontSizes": {"xs": "0.75rem", "sm": "0.875rem", "md": "1rem", "lg": "1.125rem", "xl": "1.25rem"}}` (기본값)
  - **AI 지침**: AI는 `theme.fontSizes.md`가 약 16px (1rem)에 해당하도록 해야 합니다. 이는 본문 텍스트 가독성을 위한 널리 받아들여지는 표준입니다. 다른 크기는 조화로운 타이포그래피 스케일을 만들기 위해 비례적으로 설정해야 합니다.

- `theme.lineHeights` (dict): 미리 정의된 줄 높이 토큰 (일반적으로 'xs', 'sm', 'md', 'lg', 'xl')의 스케일을 정의합니다.
    - 예시: `theme={"lineHeights": {"xs": 1.4, "sm": 1.45, "md": 1.55, "lg": 1.6, "xl": 1.65}}` (기본값)
    - **AI 지침:** AI는 `theme.lineHeights.md` (표준 본문 텍스트의 경우)를 권장되는 대로 1.5에서 1.6 사이의 값으로 설정하여 최적의 가독성을 보장해야 합니다.

2.3. 간격, 곡선 및 그림자

일관된 간격, 적절한 모서리 둥글게 처리, 미묘한 그림자 사용은 깔끔하고 정리되어 있으며 탐색하기 쉬운 UI에 크게 기여합니다.

간격, 곡선 및 그림자 원칙:

- **여백 (Whitespace):** 네거티브 공간이라고도 불리는 여백은 디자인 요소가 숨쉴 공간을 제공하는 데 중요합니다. 가독성을 높이고, 사용자 초점을 유도하며, 콘텐츠를 분할하는 데 도움이 됩니다. 기본 단위 (예: 4px 또는 8px)의 배수를 기반으로 하는 잘 정의된 간격 시스템을 설정하고 일관되게 사용해야 합니다.
- **깊이 및 질감:** 그림자는 깊이감을 형성하여 대화형 요소를 더 눈에 띄게 만들고 시각적 계층 구조를 설정하는 데 사용될 수 있습니다. 일반적으로 날카로운 그림자보다 부드러운 그림자가 선호됩니다. 배경색과 그림자색을 일치시키면 응집력을 높일 수 있습니다. 일관된 그림자 시스템 (예: 다른 높이에 대해 다른 그림자 강도)을 정의해야 합니다.
- **모양 일관성:** 버튼, 카드, 입력 요소와 같은 요소 전반에 걸쳐 모서리 반경의 일관성을 유지하는 것은 조화롭고 전문적인 모양에 기여합니다.

간격, 곡선 및 그림자를 위한 DMC 구현:

- `theme.spacing` (dict): 마진, 패딩 및 레이아웃 컴포넌트의 간격에 사용되는 간격 토큰 (예: 'xs', 'sm', 'md', 'lg', 'xl')의 스케일을 정의하는 사전입니다. 이러한 토큰은 일반적으로 픽셀 또는 rem 값에 매핑됩니다.
    - 예시: `theme={"spacing": {"xs": "0.25rem", "sm": "0.5rem", "md": "1rem", "lg": "1.5rem", "xl": "2rem"}}` (기본 글꼴 크기가 16px이면 1rem = 16px)
    - **AI 지침:** AI는 간격 시스템을 설정하는 데 권장되는 대로 4px 또는 8px와 같은 기본 단위의 배수를 기반으로 `theme.spacing`을 일관된 스케일로 정의해야 합니다. 그런 다음 이러한 토큰은 다양한 컴포넌트의 `m` (마진), `p` (패딩)와 `dmc.Stack`, `dmc.Group`, `dmc.Grid`와 같은 레이아웃 컴포넌트의 `gap` 또는 `gutter` 속성에 사용되어야 합니다.
- `theme.radius` (dict) 및 `theme.defaultRadius` (string): `theme.radius`는 테두리 반경 토큰 (예: 'xs', 'sm', 'md', 'lg', 'xl')의 스케일을 정의하고, `theme.defaultRadius`는 재정의되지 않는 한 대부분의 컴포넌트에 적용될 기본 테두리 반경 값 (`theme.radius`의 키)을 설정합니다.
    - 예시: `theme={"radius": {"sm": "0.25rem", "md": "0.5rem", "lg": "1rem"}, "defaultRadius": "md"}`
    - **AI 지침:** AI는 `defaultRadius`를 설정하여 애플리케이션 전체에서 모서리 둥글게 처리의 전반적인 일관성을 보장해야 합니다. 이러한 반경 토큰은 `dmc.Card`, `dmc.Button`, `dmc.Input`, `dmc.Paper`와 같은 컴포넌트에 각각의 `radius` 속성을 통해 적용되어야 합니다. 이는 모양 및 모서리 반경 일관성 원칙을 직접적으로 지원합니다.
- `theme.shadows` (dict): 깊이의 환상을 만들기 위해 컴포넌트에 적용될 수 있는 박스-그림자 토큰 (예: 'xs', 'sm', 'md', 'lg', 'xl')의 스케일을 정의하는 사전입니다.
    - 예시: `theme={"shadows": {"sm": "0 1px 3px rgba(0,0,0,0.05), 0 1px 2px rgba(0,0,0,0.1)", "md": "0 4px 6px -1px rgba(0,0,0,0.1), 0 2px 4px -1px rgba(0,0,0,0.06)"}}` (Mantine의 기본 그림자 값)

- **AI 지침:** AI는 `theme.shadows` 내에 미묘하고 일관된 그림자 시스템을 정의해야 합니다. 그런 다음 이러한 토큰은 `dmc.Card(shadow="sm")` 또는 `dmc.Paper(shadow="md")`와 같은 컴포넌트에 적용되어 시각적 깊이와 계층 구조를 생성할 수 있습니다. AI는 더 부드러운 그림자를 사용하도록 안내되어야 합니다. Mantine의 기본 그림자는 일반적으로 중립적이지만, 매우 색상이 있는 배경이 사용되는 경우, 그림자 정의 자체에 배경과 일치하는 색조를 포함해야 할 수 있습니다.

2.4. 전역 컴포넌트 스타일링

UI 디자인에서 일관성과 단순성을 달성하는 것은 컴포넌트에 대한 전역 스타일 또는 기본 속성을 설정하는 능력에 크게 도움이 됩니다. 이는 버튼이나 카드와 같은 요소가 애플리케이션 전체에서 반복적인 스타일링 없이 균일한 모양과 동작을 갖도록 보장합니다.

전역 스타일링 원칙:

- **일관성:** UI 요소는 애플리케이션의 모든 부분에서 유사한 방식으로 작동하고 표시되어야 합니다. 이는 사용자 신뢰를 구축하고, 인터페이스를 예측 가능하게 만들어 인지 부하를 줄이며, 응집력 있는 미학에 기여합니다.
- **단순성:** 디자인은 깔끔해야 하며, 필수 기능 및 정보를 우선시해야 합니다. 컴포넌트 스타일링의 불필요한 변형을 줄이는 것은 이러한 단순성에 기여합니다.

`theme.components` (dict)를 통한 DMC 구현: `MantineProvider`의 `theme` 객체에 있는 `theme.components` 키는 전역 컴포넌트 사용자 정의를 위한 강력한 메커니즘입니다. 이를 통해 모든 DMC 컴포넌트에 대한 기본 속성 및 기본 스타일을 지정할 수 있습니다.

- **`defaultProps` 설정:** 이를 통해 AI는 모든 DMC 컴포넌트의 속성에 대한 기본값을 정의할 수 있습니다. 이 속성이 명시적으로 설정되지 않은 상태로 컴포넌트가 사용될 때, 테마의 기본값이 적용됩니다.
  - 예시:

```
theme = {
    "components": {
        "Button": {
            "defaultProps": {
                "variant": "filled",
                "color": "primary", # primaryColor가 정의되어 있다고
가정
                "radius": "md"      # 'md'가 theme.radius의 키라고
가정
            }
        },
        "Card": {
            "defaultProps": {
                "shadow": "sm",     # 'sm'이 theme.shadows의 키라고
가정
                "withBorder": True,
                "padding": "lg"     # 'lg'가 theme.spacing의 키라고
가정
            }
        }
    }
}
```

    - **AI 지침:** 일관성과 단순성을 적용하기 위해 AI는 `dmc.Button`, `dmc.Card`, `dmc.TextInput`, `dmc.Title`, `dmc.Text`와 같이 자주 사용되는 컴포넌트에 대한 `defaultProps`를 정의해야 합니다. 예를 들어, 모든 버튼에 대한 기본 변형 및 반경을 설정하거나, 모든 카드에 대한 기본 그림자 및 패딩을 설정하면 균일한 기본 모양과 느낌이 보장됩니다. 이는 AI가 각 컴포넌트 인스턴스에 대해 생성해야 하는 코드 양을 줄이고, 한 곳에서 테마를 수정하여 전역 스타일 변경을 더 쉽게 만듭니다.
- **전역 `styles` 또는 `classNames` 적용 (Styles API):** `defaultProps` 외에도 `theme.components` 키는 Mantine의 Styles API를 활용하여 컴포넌트의 특정 내부 선택기에 전역 스타일 또는 CSS 클래스 이름을 적용하는 데 사용될 수 있습니다. 이는 컴포넌트 부분의 모양을 더 세밀하게 제어할 수 있도록 합니다.
    - 예시:

```
theme = {
    "spacing": {"sm": "0.5rem"}, # 아래에서 사용하기 위해 정의
    "colors": {"gray": ["#F8F9FA", ..., "#373A40"]}, # 아래에서 사용
하기 위해 정의
    "components": {
        "Title": {
            # 동적 스타일 또는 직접 사전 적용
            "styles": lambda theme, props: {
                "root": {"marginBottom": theme["spacing"]["sm"]}
            }
        },
        "TextInput": {
            # 동적 스타일 또는 직접 사전 적용
            "styles": lambda theme, props: {
                "input": {"borderColor": theme["colors"]["gray"]
[4]}
            }
        }
    }
}
```

    - **AI 지침:** 컴포넌트의 특정 내부 부분의 전역 스타일링 (예: 모든 `dmc.TextInput` 입력 요소의 테두리 색상을 균일하게 변경하거나, 모든 `dmc.Title` 컴포넌트에 일관된 아래쪽 여백 추가)의 경우, AI는 `theme.components.ComponentName.styles`를 사용해야 합니다. `styles` 값은 사전 또는 테마 및 컴포넌트 속성을 받아들이는 함수가 될 수 있으며, 스타일 사전을 반환합니다. AI는 각 컴포넌트에 사용 가능한 선택기 (예: `dmc.TextInput`의 `root`, `input`, `label`; `dmc.Button`의 `root`, `label`, `inner`)를 찾으려면 개별 DMC 컴포넌트 문서를 참조해야 합니다.

`theme.components` 키는 AI 기반 UI 생성에 특히 효과적인 도구입니다. "합리적인 기본값"을 설정함으로써 AI가 각 컴포넌트 인스턴스에 대해 생성해야 하는 보일러플레이트 코드를 크게 줄입니다. 이는 더 깔끔하고 유지보수 가능한 코드를 만들 뿐만 아니라 본질적으로 일관된 UI를 촉진하여 "깔끔하고 유용한" 애플리케이션에 대한 사용자 요청을 직접적으로 해결합니다.

표 2.1: 핵심 `dmc.MantineProvider` 테마 속성

테마 키	설명	예시 값 (설명)	관련 PDF 원칙

테마 키	설명	예시 값 (설명)	관련 PDF 원칙
`primaryColor`	애플리케이션의 주요 강조 색상을 설정합니다. `theme.colors`의 키여야 합니다.	`"blue"`, `"myCustomGreen"`	색상 (p.65), 대비 (p.20), 시각적 계층 구조 (p.7)
`colors`	사용자 정의 색상 팔레트를 정의하거나 기존 색상을 재정의합니다. 각 색상은 10가지 음영의 배열입니다.	`{"myBrandBlue": ["#E7F5FF", ..., "#1864AB"]}`	색상 (p.65-70), 대비 (p.21), 일관성 (p.37)
`primaryShade`	밝은 및 어두운 모드에 사용되는 `primaryColor`의 음영을 지정합니다.	`6` 또는 `{"light": 6, "dark": 8}`	색상 (p.65), 대비 (p.21)
`fontFamily`	대부분의 컴포넌트에 대한 전역 글꼴 모음.	`"Inter, sans-serif"`	타이포그래피 (p.71-74, 가독성)
`headings`	`dmc.Title` (h1-h6)에 대한 전역 스타일 (글꼴 모음, 글꼴 두께, 크기 (글꼴 크기, 줄 높이 수준별) 포함).	`{"fontFamily": "Roboto", "sizes": {"h1": {"fontSize": "2.5rem"}}}`	타이포그래피 (p.71, p.83, p.87-92, 시각적 계층 구조, 가독성)
`fontSizes`	미리 정의된 글꼴 크기 토큰 (xs, sm, md, lg, xl).	`{"md": "1rem"}`	타이포그래피 (p.89, 가독성, 시각적 계층 구조)
`lineHeights`	미리 정의된 줄 높이 토큰 (xs, sm, md, lg, xl).	`{"md": 1.55}`	타이포그래피 (p.90-92, 가독성)
`spacing`	마진, 패딩, 간격에 대한 간격 토큰 (xs, sm, md, lg, xl)을 정의합니다.	`{"md": "1rem"}`	여백 (p.24-29), 근접성 (p.12), 정렬 (p.16)
`radius`	테두리 반경 토큰 (xs, sm, md, lg, xl)을 정의합니다.	`{"md": "0.5rem"}`	일관성 (p.38, 모양/모서리 반경), 깊이 & 질감
`defaultRadius`	대부분의 컴포넌트에서 사용되는 `theme.radius`의 기본 테두리 반경 키.	`"md"`	일관성 (p.38, 모양/모서리 반경)
`shadows`	박스-그림자 토큰 (xs, sm, md, lg, xl)을 정의합니다.	`{"sm": "0 1px 3px rgba(0,0,0,0.1)"}`	깊이 & 질감 (p.49-55), 시각적 계층 구조
`components`	특정 DMC 컴포넌트에 대한 전역 `defaultProps` 또는 `styles`를 설정할 수 있도록 합니다.	`{"Button": {"defaultProps": {"size": "lg"}}}`	일관성 (p.37-45), 단순성 (p.22), 상호작용 비용 (상용구 감소)

테마 키	설명	예시 값 (설명)	관련 PDF 원칙
`other`	`theme["other"]["myCustomValue"]`를 통해 접근할 수 있는 사용자 정의 속성을 저장하는 사전입니다.	`{"myCustomValue": "any data"}`	확장성, 사용자 정의 로직
`black`, `white`	테마의 기본 검정색 및 흰색을 정의합니다.	`"#000000"`, `"#FFFFFF"`	색상 (p.67, p.82), 대비
`autoContrast`	`True`인 경우, 색상 배경에서 더 나은 대비를 위해 텍스트 색상을 자동으로 조정합니다.	`True` 또는 `False`	대비 (p.21), 접근성, 가독성
`luminanceThreshold`	`autoContrast`가 텍스트가 밝거나 어두워야 하는지 결정하는 데 사용하는 임계값.	`0.3` (기본값)	대비 (p.21), 접근성
`fontFamilyMonospace`	`dmc.Code`와 같은 모노스페이스 요소의 글꼴 모음.	`"ui-monospace, SFMono-Regular, ..."`	타이포그래피 (p.71, 코드 가독성)
`breakpoints`	em 단위로 반응형 중단점 (xs, sm, md, lg, xl)을 정의합니다.	`{"sm": "48em"}`	레이아웃 (반응형 디자인), 장치 간의 명확성 및 유용성과 간접적으로 관련.

`dmc.MantineProvider` 및 해당 `theme` 객체를 통한 테마에 대한 이러한 구조화된 접근 방식은 AI가 시각적으로 매력적이고 브랜드에 부합할 뿐만 아니라 핵심 UI/UX 원칙을 본질적으로 구현하여 더 깔끔한 코드와 더 유용하고 일관된 사용자 경험을 제공하는 Dash 애플리케이션을 생성할 수 있도록 지원합니다.

3. Dash Mantine Components의 레이아웃 및 구조

효과적인 레이아웃은 미학적으로 보기 좋고 매우 유용한 인터페이스를 만드는 데 기본입니다. 이는 사용자를 안내하고, 명확한 계층 구조를 설정하며, 정보를 조직화되고 접근 가능한 방식으로 제시하기 위해 요소를 전략적으로 배열하는 것을 포함합니다. Dash Mantine Components (DMC)는 디자인 원칙과 함께 사용될 때 잘 구조화되고 반응형 애플리케이션을 생성할 수 있는 다양한 레이아웃 컴포넌트 세트를 제공합니다.

3.1. 시각적 계층 구조 달성

시각적 계층 구조는 요소가 상대적 중요도를 전달하기 위해 배열되는 방식을 결정하여 사용자 주의 및 행동을 안내합니다. 이는 크기, 색상, 두께 및 위치를 조작하여 달성됩니다. 중요한 정보를 우선시하고 덜 중요한 요소에 부당한 중요성을 부여하지 않는 것이 중요합니다.

시각적 계층 구조를 위한 DMC 구현:

- **타이포그래피:** 텍스트 기반 계층 구조를 설정하는 주요 도구는 `dmc.Title` 및 `dmc.Text`입니다.
  - `dmc.Title`: `order` 속성 (1-6)과 함께 사용하여 의미론적 제목 수준을 정의합니다. `size` 속성 (테마 키 'h1'-'h6', 'xs'-'xl' 또는 숫자 값 허용) 및 `fw` 속성 (예: 400, 700, 'bold')은 시각적 중요도를 미세 조정할 수 있도록 합니다.
  - `dmc.Text`: 단락 및 기타 텍스트 콘텐츠에 `size`, `fw`, `c` 속성을 사용합니다.

    - **AI 지침:** 메인 페이지 또는 섹션 제목의 경우, AI는 낮은 `order` 값 (예: `order=1` 또는 `order=2`)과 눈에 띄는 크기를 가진 `dmc.Title`을 사용해야 합니다. 부제목 및 덜 중요한 텍스트는 `dmc.Title`을 더 높은 `order` 값과 함께 사용하거나, 더 작은 `size` 및 잠재적으로 `c="dimmed"` 또는 `theme.colors`의 더 밝은 회색 음영을 가진 `dmc.Text`를 사용하여 강조를 줄이면서 좋은 대비를 보장해야 합니다.
- **컴포넌트 중요도:** 컴포넌트의 선택 및 스타일링은 계층 구조에 크게 영향을 미칠 수 있습니다.
    - `dmc.Button`: 주요 Call-to-Action (CTA)은 시각적으로 구별되어야 합니다. 기본 버튼에는 `variant="filled"` 및 `color=theme.primaryColor`를 사용합니다. 보조 또는 삼차 작업은 버튼 계층 구조 원칙에서 설명된 대로 덜 눈에 띄게 나타나도록 `variant="light"`, `variant="outline"`, 또는 `variant="subtle"`을 사용할 수 있습니다.
    - `dmc.Card`: `shadow` 속성은 중요한 카드를 올릴 때 사용하여 사용자에게 더 가깝게 보이도록 하여 더 중요하게 만들 수 있습니다.
- **색상 및 대비:** 테마에서 논의된 바와 같이, `theme.primaryColor`는 주요 대화형 요소에만 사용해야 합니다. 텍스트는 항상 배경과 충분한 대비를 가져야 가독성을 보장하고 적절하게 주의를 끌 수 있습니다.
- **크기 및 위치:** 더 큰 요소와 눈에 띄는 위치 (예: 페이지 상단, 중앙)에 배치된 요소는 사용자 주의를 더 많이 끄는 경향이 있습니다. DMC의 레이아웃 컴포넌트 (`dmc.Grid`, `dmc.Stack`, `dmc.Group`)는 의도된 시각적 계층 구조를 강화하기 위해 요소를 전략적으로 배치하는 데 중요합니다.

3.2. 그리드 시스템 및 반응형 레이아웃

그리드 시스템은 정렬되고 일관성 있으며 조직화된 레이아웃을 만드는 데 필수적입니다. 이들은 시각적 순서를 인터페이스의 다른 부분에 걸쳐 유지하고 다양한 화면 크기에 적응하는 데 도움이 되는 구조적 기반을 제공합니다.

그리드 시스템을 위한 DMC 구현:

- **`dmc.Grid` 및 `dmc.GridCol`:** 이 컴포넌트들은 기본적으로 12개 열을 기반으로 하는 유연한 열 기반 레이아웃 시스템을 구현합니다.
    - `dmc.Grid` 속성:
        - `columns` (number): 그리드의 총 열 수를 정의합니다 (기본값은 12).
        - `gutter` (string | number): 열 사이의 간격을 설정합니다. 테마 간격 키 (예: "xs", "sm", "md", "lg", "xl") 또는 숫자 픽셀 값을 허용합니다.
        - `justify` (string): 그리드 내 열의 가로 정렬을 제어합니다 (예: 'flex-start', 'center', 'space-between').
        - `align` (string): 그리드 내 열의 세로 정렬을 제어합니다 (예: 'stretch', 'center', 'flex-start').
        - `grow` (boolean): `True`인 경우, 마지막 행의 열이 사용 가능한 공간을 채우기 위해 확장됩니다.
    - `dmc.GridCol` 속성:
        - `span` (number | dict): `dmc.Grid.columns`에 정의된 총 열 수 중에서 `GridCol`이 차지해야 하는 열 수를 지정합니다. 반응형 레이아웃의 경우, 이 속성은 중단점 이름 (예: 'base', 'xs', 'sm', 'md', 'lg', 'xl')을 키로 하고 해당 중단점 이상에서의 열 스팬을 값으로 하는 사전을 사용할 수 있습니다. 예시: `span={{ "base": 12, "sm": 6, "lg": 4 }}`는 열이 매우 작은 화면에서는 전체 너비를, 작은 화면에서는 절반 너비를, 큰 화면에서는 3분의 1 너비를 차지함을 의미합니다.
        - `offset` (number | dict): `GridCol`을 배치하기 전에 지정된 수의 열을 건너뜁니다. 반응형 사전 형식도 지원합니다.
        - `order` (number | dict): 열의 시각적 순서를 변경합니다. 반응형 사전 형식도 지원합니다.
    - **AI 지침:** AI는 페이지의 주요 콘텐츠 영역을 구조화하는 데 `dmc.Grid`를 사용해야 합니다. 일관된 간격을 위해 `theme.spacing` 키 (예: `gutter="md"`)를 사용하여 `gutter`를 정의해야 합니다. `dmc.GridCol` 컴포넌트의 경우, AI는 콘텐츠가 다양한 화면 크기에서 적절하게 재배치되고 유용성을 유지하도록 반응형 `span` 속성 (및 필요한 경우 `offset`/`order`)을 광범위하게 사용해야 합니다.

- **`dmc.SimpleGrid`**: 이 컴포넌트는 사용 가능한 공간 내에서 각 항목이 자동으로 동일한 너비를 차지하는 반응형 그리드 레이아웃을 만드는 데 사용됩니다. 이러한 시나리오에서는 `dmc.Grid`보다 구성하기가 더 간단합니다.
    - 속성:
        - `cols` (number | dict): 열 수를 정의합니다. 반응형 사전 형식 (예: `cols={{ "base": 1, "sm": 2, "md": 3 }}`)을 지원하여 화면 너비에 따라 열 수를 변경합니다.
        - `spacing` (string | number | dict): 테마 간격 키 또는 숫자 값을 사용하여 그리드 항목 사이의 가로 및 세로 간격을 모두 설정합니다. 반응형 사전 형식도 지원합니다.
        - `verticalSpacing` (string | number | dict): 세로 간격을 독립적으로 설정하여 세로 축에 대한 `spacing`을 재정의합니다. 반응형 사전 형식도 지원합니다.
    - **AI 지침:** AI는 `dmc.SimpleGrid`를 사용하여 `dmc.Card` 컴포넌트 시리즈, 기능 강조 표시 또는 이미지 갤러리와 같이 동일한 크기의 항목이 필요한 레이아웃에 사용해야 합니다. 반응형 `cols` 및 `spacing` 속성은 효과의 핵심입니다.

`dmc.GridCol` (`span`), `dmc.SimpleGrid` (`cols`), `dmc.Flex` (`direction`, `gap`)와 같은 레이아웃 컴포넌트의 반응형 속성과 테마 정의 중단점 (액세스 가능하지만 일반적으로 이러한 속성에 의해 암묵적으로 처리됨)의 조합은 진정으로 적응 가능한 UI를 만드는 데 기본입니다. 이는 레이아웃이 데스크톱 환경에서 잘 구조화될 뿐만 아니라 더 작은 화면에서도 사용 가능하고 가독성을 유지하여 "깔끔하고 유용한" 애플리케이션 요구 사항에 직접적으로 기여합니다.

3.3. 요소 쌓기 및 그룹화

관련 요소를 올바르게 그룹화하고 정렬하는 것은 직관적이고 시각적으로 질서 있는 인터페이스를 만드는 데 필수적입니다. 근접성 원칙은 함께 배치된 요소가 관련되어 있다고 인식되는 반면, 정렬은 시각적 조화를 보장합니다.

요소 쌓기 및 그룹화를 위한 DMC 구현:

- **`dmc.Stack`**: 자식 컴포넌트를 세로 Flex 컨테이너에 배열합니다.
    - 속성:
        - `gap` (string | number | dict): 쌓인 항목 사이의 간격을 정의합니다. 테마 간격 키 (예: "sm", "md") 또는 숫자 픽셀 값을 허용합니다. 반응형 사전 형식도 지원합니다.
        - `align` (string): 교차 축 (세로 스택의 경우 가로)을 따라 항목의 정렬을 제어합니다 (예: 'stretch', 'center', 'flex-start', 'flex-end').
        - `justify` (string): 주 축 (세로)을 따라 항목의 정렬을 제어합니다 (예: 'flex-start', 'center', 'space-between').
    - **AI 지침:** AI는 양식 (레이블-입력 쌍), 항목 목록 또는 컨트롤 뒤에 오는 텍스트 섹션과 같이 요소를 세로로 배열하는 데 `dmc.Stack`을 사용해야 합니다. 디자인 원칙에 따라 일관된 세로 리듬과 적절한 여백을 유지하기 위해 `theme.spacing` 키 (예: `gap="sm"`)를 사용하여 `gap` 속성을 설정해야 합니다.
- **`dmc.Group`**: 자식 컴포넌트를 가로 Flex 컨테이너에 배열합니다.
    - 속성:
        - `gap` (string | number | dict): 그룹화된 항목 사이의 간격을 정의하며, 테마 간격 키 또는 숫자 값을 허용합니다. 반응형 사전 형식도 지원합니다.
        - `align` (string): 그룹 내 항목의 세로 정렬을 제어합니다 (예: 'center', 'flex-start'). 기본값은 'center'입니다.
        - `justify` (string): 항목의 가로 정렬을 제어합니다 (예: 'flex-start', 'center', 'space-between'). 기본값은 'flex-start'입니다.
        - `grow` (boolean): `True`인 경우, 자식 요소는 사용 가능한 가로 공간을 채우기 위해 확장하려고 시도합니다.

        - `wrap` (string): 컨테이너 너비를 초과할 경우 항목이 줄 바꿈되는 방식을 제어합니다 (예: 'wrap', 'nowrap'). 기본값은 'wrap'입니다.
    - **AI 지침:** AI는 버튼 그룹, 인라인 양식 요소 또는 아이콘-텍스트 페어링과 같이 요소를 가로로 배열하는 데 `dmc.Group`을 활용해야 합니다. `gap` 속성은 근접성 및 여백 원칙을 준수하면서 일관된 간격을 유지하는 데 중요합니다.
- **`dmc.Flex`**: 이 컴포넌트는 Flexbox 레이아웃을 포괄적으로 제어할 수 있으며, `dmc.Stack` 또는 `dmc.Group`이 충분한 유연성을 제공하지 않을 때 가로 및 세로 배열 모두에 적합합니다.
    - 속성:
        - `direction` (string | dict): Flex 방향을 설정합니다 (예: 'row', 'column', 'row-reverse', 'column-reverse'). 반응형 사전 형식 (예: `direction={{"base": "column", "sm": "row"}}`)을 지원합니다.
        - `wrap` (string | dict): Flex 줄 바꿈을 제어합니다 (예: 'wrap', 'nowrap'). 반응형 사전 형식도 지원합니다.
        - `align` (string | dict): `align-items`를 제어합니다. 반응형 사전 형식도 지원합니다.
        - `justify` (string | dict): `justify-content`를 제어합니다. 반응형 사전 형식도 지원합니다.
        - `gap` (string | number | dict): 테마 간격 키 또는 숫자 값을 사용하여 Flex 항목 사이의 간격을 설정합니다. 반응형 사전 형식도 지원합니다.
        - `rowGap`, `columnGap`: 독립적인 행 및 열 간격에 사용됩니다.
    - **AI 지침:** 미묘한 Flexbox 제어 또는 반응형 방향 변경이 필요한 더 복잡한 1차원 레이아웃의 경우, AI는 `dmc.Flex`를 사용해야 합니다. 적응형 레이아웃을 만들기 위해 반응형 속성을 광범위하게 활용해야 합니다.

`dmc.Grid`의 `gutter` 속성, `dmc.SimpleGrid`의 `spacing`, `dmc.Stack`, `dmc.Group`, `dmc.Flex`의 `gap`을 통해 `theme.spacing` 토큰을 일관되게 적용하는 것이 가장 중요합니다. 이 관행은 "여백" 및 "근접성" 원칙을 체계적으로 적용합니다. AI에게 임의의 픽셀 값 대신 이러한 테마 키 (예: `gutter="md"`, `gap="lg"`)를 사용하도록 지시함으로써, 결과 UI는 더 나은 시각적 리듬, 관련 요소의 논리적 그룹화, 충분한 여유 공간을 보여주어 더 조화롭고 쉽게 스캔 가능한 인터페이스를 제공합니다.

3.4. 간격 및 포함

공간의 전략적 사용과 콘텐츠의 효과적인 포함은 균형 잡히고 가독성이 좋으며 시각적으로 매력적인 인터페이스를 만드는 데 필수적입니다. 여백은 요소가 "숨쉴" 공간을 제공하는 반면, 컨테이너는 콘텐츠 너비를 관리하고 초점을 맞추는 데 도움이 됩니다.

간격 및 포함을 위한 DMC 구현:

- **`dmc.Space`**: 이 유틸리티 컴포넌트는 명시적인 가로 (`w` 속성) 또는 세로 (`h` 속성) 공간을 추가합니다. `h` 및 `w`의 값은 테마 간격 키 (예: "md", "xl") 또는 숫자 픽셀 값일 수 있습니다.
    - **AI 지침:** AI는 다른 컴포넌트의 마진 또는 패딩 속성이 적합하지 않거나 충분하지 않은 경우 `dmc.Space(h="md")` 또는 `dmc.Space(w="xl")`을 사용하여 의도적인 여백을 삽입해야 합니다. 이는 충분한 공간으로 시작하여 이를 개선하는 원칙과 일치합니다.
- **`dmc.Container`**: 이 컴포넌트는 콘텐츠를 부모 내에서 가로로 중앙에 배치하고 테마를 기반으로 가로 패딩을 적용합니다. 특히 가독성을 높이기 위해 콘텐츠 섹션, 특히 텍스트의 최대 너비를 제어하는 데 중요합니다.
    - 속성:
        - `size` (string | number): 컨테이너의 최대 너비를 설정합니다. 미리 정의된 최대 너비에 해당하는 테마 중단점 키 (예: 'sm', 'md', 'lg', 'xl') 또는 숫자 픽셀 값을 허용합니다.
        - `fluid` (boolean): `True`인 경우, 컨테이너는 `size` 속성을 무시하고 사용 가능한 너비의 100%를 차지합니다.

    - **AI 지침:** AI는 주요 페이지 콘텐츠 또는 큰 텍스트 중심 섹션을 `dmc.Container`로 감싸야 합니다. 이는 최대 너비를 제어하여 과도하게 긴 텍스트 줄을 방지하고 가독성 권장 사항에 따라 가독성을 향상시키는 데 도움이 됩니다.
- **`dmc.Center`**: 자식을 자체 경계 내에서 가로 및 세로로 모두 중앙에 배치하는 간단한 유틸리티 컴포넌트입니다.
    - 속성: `inline` (boolean): `True`인 경우, 표시를 위해 `inline-flex`를 사용하여 텍스트 줄 내에서 또는 다른 인라인 요소와 함께 중앙에 배치될 수 있도록 합니다.
    - **AI 지침:** AI는 로딩 표시기 (`dmc.Loader`), 섹션의 단일 버튼, 자리 표시자 텍스트 또는 아이콘과 같은 요소의 간단한 중앙 정렬에 `dmc.Center`를 사용해야 합니다.
- **`dmc.AspectRatio`**: 이 컴포넌트는 자식 콘텐츠에 대해 일관된 너비-높이 비율을 유지하며, 미디어 요소에 특히 유용합니다.
    - 속성: `ratio` (number): 종횡비를 정의합니다 (`width / height`로 계산됨. 예: 와이드스크린 비디오의 경우 16/9, 정사각형의 경우 1/1).
    - **AI 지침:** AI는 비디오 (`html.Iframe`), 이미지 (`dmc.Image`) 또는 시각적 무결성을 위해 특정 종횡비 유지가 중요한 기타 콘텐츠와 같은 요소를 포함할 때 `dmc.AspectRatio`를 사용해야 합니다.

3.5. 애플리케이션 셸

주요 헤더, 탐색 영역 및 콘텐츠 표시 영역을 포함하는 일관된 애플리케이션 구조는 유용성에 핵심이며 사용자가 애플리케이션 내에서 자신을 찾는 데 도움이 됩니다.

애플리케이션 셸을 위한 DMC 구현:

- **`dmc.AppShell`**: 이 컴포넌트는 일반적인 애플리케이션 레이아웃을 위한 기성 구조를 제공합니다. 헤더, 내비게이션 바 (사이드바), 푸터 및 선택적 보조 패널과 같은 일반적인 섹션을 모두 기본 콘텐츠 영역 주위에 배치합니다.
    - 자식: `dmc.AppShell`은 일반적으로 `dmc.AppShellHeader`, `dmc.AppShellNavbar`, `dmc.AppShellMain` (주요 콘텐츠용), `dmc.AppShellAside`, `dmc.AppShellFooter`와 같은 자식 컴포넌트를 포함합니다.
    - 주요 `dmc.AppShell` 속성:
        - `padding` (string | number): `AppShellMain` 섹션의 패딩을 제어하며, 테마 간격 키 또는 숫자 값을 허용합니다.
        - `layout` (string): 'default' 또는 'alt'일 수 있으며, Navbar/Aside가 Header/Footer와 관련하여 배열되는 방식에 영향을 줍니다.
        - `zIndex` (number | string): 셸 요소의 z-인덱스를 설정합니다.
        - `withBorder` (boolean): 관련 컴포넌트 (Header, Navbar 등)에 테두리가 있어야 하는지 여부를 결정합니다.
        - `header`, `navbar`, `aside`, `footer` (dicts): 이 속성들은 중요합니다. 각각 크기 (`height`는 Header/Footer, `width`는 Navbar/Aside), 반응형 동작을 위한 중단점 (예: Navbar가 접히거나 숨겨져야 하는 경우), 및 접힌 상태를 구성하는 사전을 사용합니다.
        - 예시: `dmc.AppShell(header={"height": 60}, navbar={"width": 300, "breakpoint": "sm", "collapsed": {"mobile": True, "desktop": False}}, children=[...])`
    - **AI 지침:** 표준 애플리케이션 인터페이스의 경우, AI는 `dmc.AppShell`을 사용하여 일관되고 전문적인 프레임을 설정해야 합니다. 반응형 동작을 보장하기 위해 적절한 높이/너비 값과 중요한 중단점 설정을 사용하여 `header`, `navbar`, `aside` (사용된 경우), `footer` 속성을 신중하게 구성해야 합니다. 탐색 링크는 일반적으로 `dmc.App` 내에 배치되어야 합니다.

4. 데이터 표시 및 피드백

이 섹션에서는 Dash Mantine Components를 사용하여 애플리케이션에서 데이터를 표시하고 사용자에게 피드백을 제공하는 방법을 다룹니다.

4.1. 이미지 표시

이미지는 시각적 콘텐츠를 전달하는 데 필수적입니다. 효율적인 이미지 표시는 성능과 미학을 모두 고려해야 합니다.

이미지 표시를 위한 DMC 구현:

- `dmc.Image`: 이미지를 표시하기 위한 컴포넌트입니다.
  - 속성:
    - `src` (string): 이미지의 URL입니다.
    - `alt` (string): 이미지의 대체 텍스트로, 접근성에 중요합니다.
    - `fit` (string: 'cover' | 'contain' | 'fill' | 'none' | 'scale-down'): 컨테이너 내에서 이미지 크기를 조정하는 방법을 제어합니다.
    - `radius` (string | number): 이미지의 테두리 반경을 설정합니다.
    - `h`, `w` (string | number): 이미지의 높이와 너비를 설정합니다.
    - `withPlaceholder` (boolean): `True`인 경우, 이미지 `src`가 제공되지 않거나 로드에 실패하면 자리 표시자 (아이콘 및/또는 텍스트)를 표시합니다.
    - `placeholder` (Dash 컴포넌트): 자리 표시자로 표시할 사용자 정의 콘텐츠입니다.
    - `caption` (string | Dash 컴포넌트): 이미지 아래에 표시할 캡션입니다.
  - **AI 지침:** AI는 접근성을 보장하기 위해 `dmc.Image`에 대한 설명적인 `alt` 속성을 항상 제공해야 합니다. 제한된 컨테이너 (예: `dmc.CardSection` 또는 `dmc.GridCol`) 내에 이미지를 배치할 때, AI는 이미지 왜곡 없이 잘 표시되도록 `fit` 속성 ('cover' 또는 'contain')을 적절하게 사용해야 합니다. 특정 크기가 필요한 경우 `h` 및 `w` 속성을 사용해야 합니다. 일관성을 위해 다른 요소의 `defaultRadius`와 일치하도록 `radius`를 사용하는 것을 고려하십시오. 로드되지 않거나 선택 사항인 이미지의 경우 `withPlaceholder=True`를 설정하여 우아한 폴백을 제공해야 합니다. `caption` 속성은 이미지에 대한 컨텍스트 또는 출처를 제공하는 데 사용할 수 있습니다.

4.2. 목록 및 테이블

정보를 체계적으로 표시하는 데 목록과 테이블은 필수적입니다.

목록 및 테이블을 위한 DMC 구현:

- `dmc.List` 및 `dmc.ListItem`: 정렬된 목록과 순서 없는 목록을 생성하고 아이콘 지원 및 중첩 기능을 제공합니다.
  - **AI 지침:** AI는 `dmc.List` 및 `dmc.ListItem`을 사용하여 항목을 명확하게 제시해야 합니다. 계층적 데이터에는 중첩 목록을 사용하고, 필요에 따라 `icon` 속성을 사용하여 시각적 구분을 추가해야 합니다.
- `dmc.Table`: 테이블 형식 데이터를 표시하는 데 사용되며, Mantine 테마에 따라 스타일이 지정됩니다. 표준 `html.Table`의 향상된 대안 역할을 합니다. `dmc.Table`은 Mantine 테마와의 원활한 통합과 셀 내에 다른 Dash 컴포넌트를 직접 포함할 수 있는 기능을 원하는 작은 데이터 세트에 특히 적합합니다. 대규모 데이터 세트, 성능 집약적인 UI 또는 필터링, 정렬, 가상화와 같은 고급 기능의 경우 dash-ag-grid가 권장되는 솔루션으로 남아 있습니다.
  - **AI 지침:** 작은 데이터 세트에는 `dmc.Table`을 사용하고, 큰 데이터 세트에는 `dash-ag-grid`를 사용하는 것을 고려하십시오. 테이블은 가독성을 위해 깔끔하게 유지되어야 합니다.

4.3. 피드백 및 상태 표시

사용자에게 작업의 결과나 애플리케이션의 상태를 명확하게 알리는 것은 훌륭한 UX의 핵심입니다.

피드백 및 상태 표시를 위한 DMC 구현:

- `dmc.Alert`: 사용자에게 정보를 제공하거나, 경고하거나, 오류를 표시하는 데 사용됩니다.
  - 속성: `title` (string), `children` (메시지 내용), `color` (string: 테마 색상, 예: "red", "green", "blue" 또는 CSS 색상 값), `icon` (Dash 컴포넌트, 예: DashIconify), `variant` (string: 'light', 'filled', 'outline', 'transparent', 'white'), `withCloseButton` (boolean), `duration` (number, 자동 해제를 위한 밀리초).
  - **AI 지침:** AI는 양식 유효성 검사 오류, 작업 후 성공 메시지 또는 중요한 정보 경고에 `dmc.Alert`를 사용해야 합니다. 색상 규칙을 엄격히 준수해야 합니다: 오류/위험에는 `color="red"`, 성공에는 `color="green"`, 경고에는 `color="yellow"`, 정보 알림에는 `color="blue"` 또는 다른 중립/브랜드 색상을 사용해야 합니다. 명확한 `title`과 설명적인 `children` 메시지는 필수적입니다. `icon` (예: 오류의 경우 `icon=DashIconify(icon="tabler:alert-circle")`)을 포함하면 시각적 신호가 향상됩니다.
- `dmc.Badge`: 상태, 범주 또는 태그를 나타내는 데 사용되는 작고 인라인 설명자입니다.
  - **AI 지침:** `dmc.Badge`를 사용하여 짧고 시각적으로 구별되는 상태 또는 범주를 표시합니다. `color` 및 `variant` 속성을 사용하여 적절한 시각적 피드백을 제공합니다.
- `dmc.Loader`: 데이터 로딩 중임을 사용자에게 시각적으로 알리는 데 사용됩니다.
  - **AI 지침:** 비동기 작업 중에 `dmc.Loader`를 사용하여 사용자에게 애플리케이션이 응답하고 있음을 알립니다. 적절한 `size` 및 `color`를 사용하여 시각적 흐름을 방해하지 않도록 합니다.
- `dmc.Progress` 및 `dmc.RingProgress`: 작업 완료 또는 정량적 데이터에 대한 선형 또는 원형 시각적 피드백을 제공합니다.
  - **AI 지침:** 긴 작업의 진행 상황을 표시하거나 KPI를 시각화하는 데 사용합니다. `sections` 속성을 사용하여 진행률을 명확하게 나타냅니다.
- `dmc.ThemeIcon`: 테마 배경 내에 아이콘을 표시하여 시각적 신호 및 일관성을 향상시킵니다.
  - **AI 지침:** `dmc.ThemeIcon`을 사용하여 아이콘을 시각적으로 강조하고, `color` 및 `variant`를 사용하여 애플리케이션 테마와 일치시킵니다.

이러한 데이터 표시 컴포넌트를 전략적으로 사용하고 개요에 설명된 원칙을 준수함으로써, AI는 정보를 명확하고 효율적이며 매력적으로 제시하는 Dash 애플리케이션을 만들 수 있습니다.

4.x. 데이터 시각화: Dash Mantine Components의 차트 및 그래프

Dash Mantine Components는 Recharts를 기반으로 구축된 다양한 유형의 차트를 만들기 위한 전용 스위트를 제공합니다. 이러한 컴포넌트는 데이터 추세, 비교 및 분포를 시각화하여 복잡한 데이터 세트를 더 이해하기 쉽고 실행 가능하게 만드는 데 필수적입니다. 효과적인 데이터 시각화는 명확성, 정확성 및 관련성 원칙을 준수하여 차트가 사용자를 오도하지 않고 사용자 작업을 지원하도록 보장합니다.

차트 사용을 위한 일반 원칙:

- **올바른 차트 유형 선택:** 데이터와 전달하려는 통찰력에 적합한 차트 (예: 추세에는 선 차트, 비교에는 막대 차트, 비율에는 원형 차트)를 선택합니다.
- **명확성 및 단순성:** 차트를 깔끔하고 정돈되게 유지합니다. 불필요한 장식 ("차트 쓰레기")을 피합니다. 레이블, 축 및 범례는 명확하고 가독성이 있어야 합니다.
- **색상 사용:** 데이터 시리즈를 구분하거나 주요 정보를 강조하기 위해 색상을 의도적으로 사용합니다. 색상이 접근 가능하고 잘 대비되는지 확인합니다. 일관성을 위해 `theme.colors`를 활용합니다.
- **상호 작용:** 필요에 따라 더 많은 세부 정보를 제공하기 위해 도구 설명을 활용합니다. 밀집된 데이터 세트를 다루는 경우 확대/축소 및 이동과 같은 기능을 고려합니다.

- **반응성**: 차트가 다른 화면 크기에 잘 적응하는지 확인합니다.

DMC 차트 컴포넌트 개요: 모든 DMC 차트 컴포넌트는 데이터, 시리즈, 축 및 스타일링과 관련된 일부 공통 속성을 공유합니다. 이들은 원활하게 작동하도록 설계되었습니다.

- `dmc.AreaChart`: 시간 경과에 따른 정량적 변화를 보여주는 데 유용합니다. 선 그래프와 유사하지만 선 아래 영역이 채워집니다.
- `dmc.BarChart`: 다른 범주 간의 비교를 위한 표준 선택입니다.
- `dmc.LineChart`: 시간 경과에 따른 추세 또는 연속 데이터 포인트를 표시하는 데 가장 적합합니다.
- `dmc.PieChart` 및 `dmc.DonutChart`: 전체의 비율을 표시하는 데 사용됩니다. `dmc.DonutChart`는 중앙에 공간이 있어 총계 또는 다른 정보를 표시할 수 있습니다.
- `dmc.RadarChart`: 세 개 이상의 정량적 변수를 동일한 시작점에서 축에 표시하는 다변량 데이터를 표시하는 데 사용됩니다.
- `dmc.ScatterChart`: 두 변수가 서로에게 얼마나 영향을 미치는지 보여주기 위해 가로 및 세로 축에 데이터 포인트를 플로팅하는 데 사용됩니다.
- `dmc.Sparkline`: 텍스트, 테이블 또는 KPI와 함께 임베드되어 축이나 레이블 없이 추세의 빠른 시각적 표현을 제공하는 작고 간단하며 데이터 밀도가 높은 차트입니다.
- `dmc.CompositeChart`: 동일한 플롯 영역 내에서 다른 차트 유형 (예: 막대 및 선)을 결합할 수 있습니다.

모든 차트에 대한 AI 지침: AI는 차트 데이터 (`data`, `dataKey`, `series`)가 올바르게 구조화되었는지 항상 확인해야 합니다. 스타일링의 경우, 시리즈 색상에는 `theme.colors`를, 레이블 및 텍스트에는 차트 속성을 통해 `theme.fontFamily`를 우선적으로 사용해야 하며, 일반 타이포그래피 및 접근성 원칙에 따라 차트의 모든 텍스트 요소 (축 레이블, 범례, 도구 설명)에 충분한 대비와 가독성을 보장해야 합니다. 차트가 제시될 때마다, 매우 간결한 표현을 위한 스파크라인이 아닌 한, 차트가 무엇을 보여주는지 설명하는 설명적인 `dmc.Title` 또는 `dmc.Text`가 근처에 있어야 합니다.

5. Dash Mantine Components의 타이포그래피

타이포그래피는 사용자 인터페이스 디자인의 기본 요소이며, 의사소통, 가독성, 시각적 계층 구조 및 전반적인 사용자 경험에 중요한 역할을 합니다. Dash Mantine Components (DMC)는 효과적인 타이포그래피 시스템을 구현하기 위한 타이포그래피 컴포넌트 세트와 광범위한 테마 사용자 정의 옵션을 제공합니다. 이 섹션에서는 "The UI_UX Playbook"에 설명된 원칙에 따라 특정 DMC 타이포그래피 컴포넌트 및 해당 속성을 사용하는 방법을 자세히 설명합니다.

핵심 타이포그래피 원칙 (요약):

- **글꼴 선택 및 가독성**: 컨텍스트에 적합한 글꼴을 선택하고 가독성 및 미적 매력을 우선시합니다. 1-2개의 글꼴 모음으로 제한합니다.
- **계층 구조**: 크기, 두께 및 색상의 변화를 사용하여 명확한 시각적 계층 구조를 설정하고, 중요한 정보에 사용자 주의를 유도합니다.
- **줄 길이 및 높이**: 편안한 읽기를 위해 줄 길이 (데스크톱의 경우 45-75자) 및 줄 높이 (본문 텍스트의 경우 글꼴 크기의 150-160%)를 최적화합니다.
- **대비**: 텍스트와 배경 사이에 충분한 대비 (WCAG AA 4.5:1)를 보장합니다. 대비되는 배경에 순수 검정색/흰색 텍스트를 피하고 더 부드러운 회색을 선택합니다.
- **정렬**: 왼쪽에서 오른쪽으로 읽는 언어에서 가독성을 높이기 위해 긴 텍스트의 경우 왼쪽 정렬을 선호합니다.

DMC 타이포그래피 컴포넌트:

- `dmc.Title`: Mantine 테마에서 파생된 스타일로 의미론적 HTML 제목 (`<h1>`-`<h6>`)을 렌더링하는 데 사용됩니다.

    - 주요 속성:
        - `children`: 제목의 텍스트 콘텐츠.
        - `order` (number: 1-6): HTML 제목 태그를 결정합니다 (예: `order=1`은 `<h1>`을 렌더링). 이는 `size` 속성이 설정되지 않은 경우 기본 글꼴 크기에도 영향을 줍니다.
        - `size` (string | number): 기본 글꼴 크기를 재정의합니다. 테마 제목 키 (예: `theme.headings.sizes`와 일치하는 "h1", "h3"), 테마 글꼴 크기 키 (예: `theme.fontSizes`의 "xl"), 또는 숫자 픽셀/rem 값을 허용합니다.
        - `fw` (string | number): 글꼴 두께를 설정합니다 (예: 100, 400, 700, 'bold', 'normal').
        - `c` (string): 텍스트 색상을 지정합니다. 테마 색상 키 (예: "blue", "myCustomColor.7", "dimmed") 또는 유효한 CSS 색상 문자열을 허용합니다.
        - `ta` (string: 'left' | 'center' | 'right' | 'justify'): 텍스트 정렬을 제어합니다.
        - `lineClamp` (number): 지정된 줄 수 이후에 텍스트를 자르고 생략 부호를 추가합니다.
        - `transform` (string): CSS 텍스트 변환 속성 (예: 'capitalize', 'uppercase', 'lowercase').
        - `variant` (string): 특별한 스타일링에 사용될 수 있습니다 (예: `theme.defaultGradient` 또는 스타일 속성을 통해 그라데이션이 정의된 경우 'gradient').
        - 스타일 속성 (`style` 속성 바로 가기): `fz` (fontSize), `fw` (fontWeight), `c` (color), `ff` (fontFamily), `lh` (lineHeight), `ta` (textAlign).
    - **AI 지침**: AI는 모든 의미론적 페이지 및 섹션 제목에 `dmc.Title`을 사용해야 합니다. 논리적 문서 구조를 유지하기 위해 `order` 속성을 적절하게 설정해야 합니다. `size` 및 `fw` 속성은 시각적 계층 구조를 설정하는 데 중요합니다. 텍스트 색상의 경우, 더 부드러운 대비 원칙을 준수하기 위해 순수 검정색 대신 `theme.colors`의 음영 (예: `c="dark.6"` 또는 특정 브랜드 색상)을 사용하는 것을 고려하십시오. `align`은 특정 레이아웃 요구 사항에 사용되어야 하며, 가독성을 위해 기본적으로 왼쪽으로 설정됩니다.
- **`dmc.Text`**: 단락 및 일반 텍스트 콘텐츠를 렌더링하는 주요 컴포넌트입니다.
    - 주요 속성:
        - `children`: 텍스트 콘텐츠.
        - `size` (string | number): 글꼴 크기를 설정합니다. 테마 글꼴 크기 키 (예: 'xs', 'sm', 'md', 'lg', 'xl') 또는 숫자 픽셀/rem 값을 허용합니다. 기본값은 'md'입니다.
        - `fw` (string | number): 글꼴 두께를 설정합니다.
        - `c` (string): 텍스트 색상을 지정합니다. 테마 색상 키 (하위 회색으로 매핑되는 특별한 값 "dimmed" 포함) 또는 유효한 CSS 색상 문자열을 허용합니다.
        - `ta` (string: 'left' | 'center' | 'right' | 'justify'): 텍스트 정렬을 제어합니다.
        - `span` (boolean): `True`인 경우, 기본 블록 수준 `<p>` 요소 대신 인라인 `<span>` 요소로 텍스트를 렌더링합니다.
        - `inherit` (boolean): `True`인 경우, 컴포넌트는 자체 테마 기반 스타일을 적용하는 대신 부모 요소에서 글꼴 속성 (글꼴 모음, 크기, 색상 등)을 상속합니다.
        - `gradient` (dict): 텍스트 색상에 대한 그라데이션을 정의합니다. `variant="gradient"`가 설정되어야 합니다. 사전은 `from`, `to`, `deg`를 지정해야 합니다 (예: `{"from": "indigo", "to": "cyan", "deg": 45}`).
        - `variant` (string): 'text' (기본값) 또는 'gradient'일 수 있습니다.
        - `lineClamp` (number): 지정된 줄 수 이후에 텍스트를 생략 부호로 자릅니다.
        - `truncate` (boolean | string: 'start' | 'end'): `True` 또는 'end'인 경우, 넘치는 텍스트를 끝에 생략 부호로 자릅니다. 'start'인 경우, 시작 부분에서 자릅니다 (덜 일반적).
        - `inline` (boolean): 인라인 요소로 사용될 때 더 나은 중앙 정렬을 위해 줄 높이를 1로 설정합니다.
        - 스타일 속성: `fz`, `fw`, `c`, `ff`, `lh`, `ta`.

    - **AI 지침:** AI는 모든 본문 텍스트, 레이블 (입력과 관련되지 않은), 및 기타 비 제목 텍스트에 `dmc.Text`를 사용해야 합니다. 기본 `size="md"`는 테마에 정의된 16px 기본 본문 크기에 해당해야 합니다. 강조를 줄인 텍스트 또는 보조 정보의 경우, `c="dimmed"` 또는 더 밝은 회색 (예: `c="gray.7"`)을 사용해야 하며, 항상 배경과의 충분한 대비를 보장해야 합니다. `lineClamp` 또는 `truncate` 속성은 카드 또는 테이블 셀과 같이 제한된 공간에서 텍스트 오버플로를 관리하는 데 유용합니다. AI는 본문 텍스트의 경우 효과적인 줄 높이 (스타일 속성 `lh` 또는 전역 테마 설정을 통해)가 약 1.5에서 1.6 사이여야 가독성을 최대화할 수 있도록 해야 합니다. `inherit` 속성은 `dmc.Text`가 특정 타이포그래피 스타일링이 있는 부모 요소와 원활하게 혼합되어야 할 때 유용합니다.
- **`dmc.Blockquote`**: 종종 출처와 아이콘이 있는 인용된 텍스트를 표시하는 데 사용됩니다.
    - 주요 속성:
        - `children`: 인용문의 주요 내용.
        - `color` (string): 왼쪽 테두리 및 아이콘 색상 (아이콘이 제공되고 자체적으로 색상이 지정되지 않은 경우)에 사용되는 테마 색상 키.
        - `icon` (Dash 컴포넌트): 블록 인용문 옆에 표시할 선택적 아이콘으로, 일반적으로 `DashIconify` 컴포넌트입니다.
        - `cite` (string | Dash 컴포넌트): 인용문에 대한 출처 텍스트로, 일반적으로 아래에 표시됩니다.
    - **AI 지침:** AI는 `dmc.Blockquote`를 사용하여 인용된 자료 (예: 증언 또는 발췌문)를 시각적으로 구별해야 합니다. 더 나은 시각적 신호를 위해 `icon` (예: 인용 부호 아이콘)을 제공하고, 출처 표기를 위해 `cite` 속성을 사용해야 합니다. `color` 속성은 테마의 `primaryColor` 또는 다른 강조 색상으로 설정할 수 있습니다.
- **`dmc.Code`**: 인라인 또는 블록 수준 코드 스니펫을 표시하는 데 사용됩니다.
    - 주요 속성:
        - `children` (string): 표시할 코드 문자열.
        - `color` (string): 배경 (만약 `block=True`인 경우) 또는 텍스트 색상에 대한 테마 색상 키.
        - `block` (boolean): `True`인 경우, 여러 줄 코드 스니펫에 적합한 블록 수준 요소로 렌더링되며, 종종 별도의 배경을 가집니다. `False` (기본값)인 경우, 인라인 요소로 렌더링됩니다.
        - 이 컴포넌트는 기본적으로 `theme.fontFamilyMonospace`를 사용합니다.
    - **AI 지침:** AI는 컴퓨터 코드의 모든 표현에 `dmc.Code`를 사용해야 합니다. 인라인 코드 언급의 경우 `block=False`가 적절합니다. 여러 줄 코드 예시의 경우 `block=True`를 사용해야 하며, 일반적으로 더 나은 시각적 분리를 위해 배경색과 패딩을 적용합니다.
- **`dmc.Highlight`**: 더 큰 텍스트 문자열 내에서 특정 부분 문자열의 발생을 강조하는 데 사용됩니다.
    - 주요 속성:
        - `children` (string): 강조할 전체 문자열.
        - `highlight` (string | list of strings): 강조할 부분 문자열.
        - `highlightColor` (string): 강조된 텍스트 부분의 테마 색상 키 또는 CSS 색상.
        - `color` (string): 강조되지 않은 텍스트 부분의 선택적 테마 색상 키 또는 CSS 색상.
    - **AI 지침:** AI는 검색 결과 (결과 내 검색어 강조) 또는 텍스트 구절에서 특정 키워드에 주의를 기울일 때 `dmc.Highlight`를 사용해야 합니다.

6. 신규 문서화 및 업데이트된 컴포넌트 (v1.3.0)

이 섹션에서는 버전 1.3.0에서 새로 추가되거나 중요한 업데이트를 받았거나, 완전성을 위해 문서화가 추가된 필수 컴포넌트에 대해 자세히 설명합니다. 학습 및 사용 편의성을 위해 일관된 문서 구조가 적용됩니다. 이는 일반적으로 소개, 사용 예시, 주요 기능 설명, 키워드 인수 표, Styles API 정보를 포함합니다.

- **`dmc.Table` 및 `dmc.TableScrollContainer`**: `dmc.Table`은 테이블 형식 데이터를 표시하는 데 사용되며, `dmc.TableScrollContainer`는 `dmc.Table`을 스크롤 가능하게 만듭니다. 작은 데이터 세트에 적합하

며, 큰 데이터 세트에는 `dash-ag-grid`가 권장됩니다.
- **`dmc.Carousel` 및 `dmc.CarouselSlide`**: 슬라이드쇼를 표시하는 데 사용됩니다.
- **`dmc.Timeline` 및 `dmc.TimelineItem`**: 시간 순서 이벤트를 표시하는 데 사용됩니다.
- **`dmc.Stepper` 및 `dmc.StepperStep`**: 여러 단계 프로세스를 표시하는 데 사용됩니다.
- **`dmc.CodeHighlight`, `dmc.CodeHighlightTabs`, `dmc.InlineCodeHighlight`**: 코드 스니펫을 다양한 방식으로 표시하는 데 사용됩니다.
- **`dmc.SegmentedControl`**: 연결된 버튼으로 표시되는 옵션 세트에서 하나의 옵션을 선택하는 데 사용됩니다.
- **`dmc.List` 및 `dmc.ListItem`**: 정렬된 및 순서 없는 목록을 생성하는 데 사용됩니다.
- **`dmc.Center`**: 콘텐츠를 가로 및 세로로 중앙에 배치하는 레이아웃 컴포넌트입니다.
- **`dmc.Menu`, `dmc.MenuTarget`, `dmc.MenuDropdown`, `dmc.MenuItem`**: 드롭다운 메뉴를 만드는 데 사용됩니다.
- **`dmc.ActionIcon`**: 아이콘 전용 버튼에 대한 대안입니다.
- **`dmc.Highlight`**: 텍스트 내에서 부분 문자열을 강조하는 데 사용됩니다.
- **`dmc.Pagination`**: 페이지 매김된 콘텐츠를 탐색하는 데 사용됩니다.
- **`dmc.Tree`**: 계층적 데이터 구조를 표시하는 데 사용됩니다.
- **`dmc.Grid` 및 `dmc.GridCol`**: Flexbox 그리드 시스템을 사용하여 반응형 레이아웃을 만드는 데 사용됩니다.

7. 이 가이드의 유지 관리 및 모범 사례

이 가이드가 정확하고 관련성이 있으며 사용자 친화적인지 확인하려면 다음 관행을 지속적으로 유지 관리하는 것이 좋습니다.

- **버전 고정**: 이 문서는 특히 `dash-mantine-components==1.3.0`에 맞춰져 있습니다. 이 버전 관리는 가이드의 시작 부분에 명확하게 명시되어야 합니다. `dash-mantine-components` 라이브러리의 향후 업데이트는 혼란을 방지하고 개발자가 라이브러리 버전에 관련된 정보를 참조하도록 보장하기 위해 새롭고 버전별 가이드 또는 이 문서 내에 명확하게 구분된 섹션으로 이어져야 합니다.
- **공식 문서와의 일관성**: 유지 관리자는 이 가이드를 공식 `dash-mantine-components` 웹사이트 (dash-mantine-components.com)와 정기적으로 상호 참조해야 합니다. 이 관행은 새 컴포넌트, 기존 컴포넌트 속성 변경, 사용 중단 및 모범 사례 업데이트를 식별하는 데 도움이 되어 이 가이드가 신뢰할 수 있고 최신 리소스가 되도록 보장합니다.
- **커뮤니티 기여**: 가능하다면, 커뮤니티 구성원이 불일치를 보고하거나 개선 사항을 제안하거나 업데이트에 기여할 수 있는 간단한 프로세스를 구축하면 이 가이드의 품질과 완전성을 크게 향상시킬 수 있습니다. 여기에는 전용 피드백 채널 또는 기여 워크플로가 포함될 수 있습니다.
- **실용적인 예시에 중점**: 실용적이고 이해하기 쉬우며 복사-붙여넣기 가능한 코드 예시를 포함하는 것이 중요합니다. 개발자는 각 컴포넌트의 일반적인 사용 사례를 보여주는 작동 예시를 통해 가장 효과적으로 학습하는 경우가 많습니다. 예시는 간결하고 주요 기능을 설명해야 합니다.
- **Styles API 문서**: Mantine Styles API를 지원하는 컴포넌트의 경우, 이 지원을 언급할 뿐만 아니라 가능한 경우 주요 선택기 (예: 양식 컴포넌트의 root, label, input; 목록과 같은 컴포넌트의 item)를 나열해야 합니다. 가이드 내에서 일반 Styles API 설명을 참조하는 것도 유용합니다. 이러한 세부 정보를 제공하면 개발자가 `dash-mantine-components`의 모든 사용자 정의 잠재력을 활용할 수 있습니다. `styles` 및 `classNames` 속성을 사용하여 컴포넌트의 내부 요소를 대상으로 지정할 수 있는 Styles API는 맞춤형 디자인을 달성하기 위한 강력한 기능입니다.

문서 요약

전체 문서 요약 (섹션별)

- **1. 서론: AI 기반 UI/UX 우수성**: 이 문서는 AI가 Dash Mantine Components (DMC)를 사용하여 깔끔하고 유용한 Dash 애플리케이션을 개발하는 데 필요한 포괄적인 가이드입니다. "The UI_UX Playbook"의 UI/UX 모범 사례를 AI를 위한 구체적인 지침으로 전환하는 데 중점을 둡니다. UI와 UX의 시너지를 강조하며, AI가 특정 DMC 코드 패턴과 속성을 통해 좋은 UI를 달성하는 방법을 선언적으로 설명합니다.
- **2. Dash Mantine Components를 이용한 핵심 테마**: 일관된 시각적 애플리케이션을 위한 테마 시스템의 중요성을 다룹니다. `dmc.MantineProvider`를 중심으로 색상 시스템, 타이포그래피 시스템, 간격, 곡선 및 그림자 정의, 그리고 전역 컴포넌트 스타일링을 위한 Mantine의 강력한 테마 기능을 자세히 설명합니다. `theme.primaryColor`, `theme.colors`, `theme.headings`, `theme.spacing` 등을 통해 AI가 브랜드에 부합하고 접근 가능한 UI를 만들도록 안내합니다.
- **3. Dash Mantine Components의 레이아웃 및 구조**: 미학적으로 보기 좋고 사용 가능한 인터페이스를 만드는 데 필수적인 레이아웃 원칙을 설명합니다. 시각적 계층 구조 달성, `dmc.Grid` 및 `dmc.SimpleGrid`를 사용한 그리드 시스템 및 반응형 레이아웃, `dmc.Stack`, `dmc.Group`, `dmc.Flex`를 사용한 요소 쌓기 및 그룹화, `dmc.Space`, `dmc.Container`, `dmc.Center`를 사용한 간격 및 포함, 그리고 `dmc.AppShell`을 사용한 애플리케이션 셸 구조에 대해 다룹니다.
- **4. 데이터 표시 및 피드백**: 애플리케이션에서 데이터를 표시하고 사용자에게 피드백을 제공하는 방법을 설명합니다. `dmc.Image`를 통한 이미지 표시, `dmc.List` 및 `dmc.Table`을 통한 목록 및 테이블, `dmc.Alert`, `dmc.Badge`, `dmc.Loader`, `dmc.Progress` 등을 통한 피드백 및 상태 표시에 대한 DMC 컴포넌트 사용을 다룹니다.
- **4.x. 데이터 시각화: Dash Mantine Components의 차트 및 그래프**: Recharts를 기반으로 하는 DMC의 다양한 차트 컴포넌트를 소개합니다. 올바른 차트 유형 선택, 명확성, 색상 사용, 상호 작용 및 반응성과 같은 일반적인 차트 사용 원칙을 강조하며, 다양한 차트 유형 (`AreaChart`, `BarChart`, `LineChart`, `PieChart`, `RadarChart`, `ScatterChart`, `Sparkline`, `CompositeChart`)을 간략하게 설명합니다.
- **5. Dash Mantine Components의 타이포그래피**: 타이포그래피의 중요성과 DMC에서 이를 구현하는 방법을 다시 강조합니다. `dmc.Title`, `dmc.Text`, `dmc.Blockquote`, `dmc.Code`, `dmc.Highlight`와 같은 특정 타이포그래피 컴포넌트의 사용법과 해당 속성에 대한 지침을 제공합니다.
- **6. 신규 문서화 및 업데이트된 컴포넌트 (v1.3.0)**: 버전 1.3.0에서 새로 추가되거나 중요한 업데이트를 받은 주요 DMC 컴포넌트 (예: `dmc.Table`, `dmc.Carousel`, `dmc.Timeline`, `dmc.Stepper`, `dmc.CodeHighlight` 등)를 나열하고 간략히 설명합니다.
- **7. 이 가이드의 유지 관리 및 모범 사례**: 가이드의 정확성, 관련성 및 유용성을 보장하기 위한 권장 사항을 제시합니다. 버전 고정, 공식 문서와의 일관성 유지, 커뮤니티 기여 유도, 실용적인 예시 제공, Styles API 문서화를 강조합니다.

특정 섹션 추출 및 요약

2. 핵심 테마 (Core Theming) 요약

핵심 테마는 Dash 애플리케이션의 시각적 일관성과 매력을 보장하는 데 매우 중요합니다. Dash Mantine Components (DMC)는 `dmc.MantineProvider`를 통해 강력한 테마 시스템을 제공합니다. 이 컴포넌트는 애플리케이션 전체에 걸쳐 테마를 전파하고, 색상 구성표를 관리하며, 전역 스타일을 주입합니다.

주요 테마 설정은 다음과 같습니다:

- **색상 시스템**: `theme.primaryColor`로 기본 강조 색상을 설정하고, `theme.colors` 딕셔너리에 사용자 정의 색상 팔레트(각 10가지 음영 포함)를 정의합니다. `theme.primaryShade`를 조정하여 밝은/어두운 모드에서 대비를 최적화해야 합니다. `dmc.Button` 등 컴포넌트에 색상을 적용할 때 `theme.primaryColor` 또는 의미론적 색상(예: 오류는 빨간색, 성공은 녹색)을 사용하도록 AI에게 지시합니다.
- **타이포그래피 시스템**: `theme.fontFamily`로 전역 글꼴을 설정하고, `theme.fontFamilyMonospace`로 코드 글꼴을 지정합니다. `theme.headings.sizes`를 사용하여 `dmc.Title`의 계층 구조를 정의하고,

`theme.fontSizes`와 `theme.lineHeights`로 본문 텍스트의 가독성을 최적화합니다.
- **간격, 곡선, 그림자**: `theme.spacing`으로 일관된 간격(마진, 패딩) 시스템을 구축합니다. `theme.radius`와 `theme.defaultRadius`로 모서리 둥글게 처리의 일관성을 유지하고, `theme.shadows`로 미묘한 깊이감을 표현합니다.
- **전역 컴포넌트 스타일링**: `theme.components` 딕셔너리를 사용하여 자주 사용되는 DMC 컴포넌트 (`dmc.Button`, `dmc.Card` 등)에 `defaultProps`를 설정하고, `styles`를 통해 세부적인 내부 요소를 제어하여 코드 중복을 줄이고 일관성을 강화합니다.

이러한 테마 속성들을 신중하게 정의하고 적용함으로써, AI는 브랜드에 부합하고 접근성이 뛰어난 UI를 생성하며, 깔끔하고 유지보수 가능한 코드를 만들 수 있습니다.

3. 레이아웃 및 구조 (Layout and Structure) 요약

효과적인 레이아웃은 사용자 경험에 필수적이며, 정보를 체계적으로 제시하고 사용자를 안내합니다. DMC는 반응형 애플리케이션을 위한 다양한 레이아웃 컴포넌트를 제공합니다.

주요 레이아웃 전략은 다음과 같습니다:

- **시각적 계층 구조**: `dmc.Title`(`order`, `size`, `fw` 속성)과 `dmc.Text`를 사용하여 텍스트 기반의 계층 구조를 명확히 합니다. `dmc.Button`의 `variant` 및 `color`, `dmc.Card`의 `shadow`를 사용하여 컴포넌트의 중요도를 시각적으로 표현합니다.
- **그리드 시스템 및 반응형 레이아웃**: `dmc.Grid`와 `dmc.GridCol`을 사용하여 유연한 열 기반 레이아웃을 구축하고, `span` 속성의 반응형 딕셔너리 포맷을 통해 다양한 화면 크기에 적응하도록 합니다. `dmc.SimpleGrid`는 동일한 크기의 항목을 정렬할 때 더 간단한 대안으로 사용됩니다.
- **요소 쌓기 및 그룹화**: `dmc.Stack`은 요소를 수직으로, `dmc.Group`은 수평으로 정렬하여 근접성 원칙을 따릅니다. `dmc.Flex`는 더 복잡한 Flexbox 제어가 필요한 경우에 사용됩니다. 이들 컴포넌트의 `gap` 속성에 `theme.spacing` 토큰을 일관되게 사용하여 여백을 관리합니다.
- **간격 및 포함**: `dmc.Space`로 명시적 여백을 추가하고, `dmc.Container`로 콘텐츠의 최대 너비를 제어하여 가독성을 높입니다. `dmc.Center`는 요소를 중앙에 배치하는 데 사용되며, `dmc.AspectRatio`는 미디어 요소의 종횡비를 유지합니다.
- **애플리케이션 셸**: `dmc.AppShell`은 헤더, 내비게이션 바, 푸터와 같은 표준 애플리케이션 구조를 제공하여 일관된 프레임을 구축하고 반응형 동작을 위한 중단점을 설정합니다.

이러한 레이아웃 컴포넌트들을 일관되게 사용하고 `theme.spacing`과 같은 테마 속성과 결합함으로써, Dash 애플리케이션은 데스크톱뿐만 아니라 모바일 환경에서도 시각적으로 정돈되고 사용 가능한 인터페이스를 제공할 수 있습니다.

Dash Mantine Components 관련 베스트 프랙티스 정리

이 문서는 Dash Mantine Components (DMC)를 사용하여 Dash 애플리케이션을 개발할 때 AI가 따라야 할 여러 모범 사례를 제시합니다. 다음은 핵심적인 베스트 프랙티스 요약입니다.

1. **테마 시스템의 일관된 사용**:
    - **`dmc.MantineProvider` 활용**: 모든 DMC 애플리케이션은 단일 `dmc.MantineProvider`로 감싸야 합니다.
    - **전역 테마 정의**: `theme` 객체를 통해 `primaryColor`, `colors`, `fontFamily`, `headings`, `spacing`, `radius`, `shadows` 등을 일관되게 정의하여 애플리케이션 전반의 시각적 통일성을 확보합니다.
    - **`defaultProps` 및 `styles`**: `theme.components`를 활용하여 자주 사용되는 컴포넌트의 `defaultProps`를 설정하고, `styles`를 통해 특정 내부 요소에 전역 스타일을 적용하여 코드 중복을 줄

이고 일관된 UI를 유지합니다.
    - **색상 대비 및 접근성:** `primaryShade`를 조정하고 `autoContrast`를 활성화하여 텍스트와 배경 간의 충분한 대비를 보장하여 가독성 및 접근성을 높입니다.
    - **의미론적 색상 활용:** `dmc.Alert` 등에서 오류는 빨간색, 성공은 녹색, 경고는 노란색과 같이 의미론적 색상 규칙을 엄격히 준수합니다.

2. **레이아웃 및 구조의 체계화:**

    - **시각적 계층 구조 명확화:** `dmc.Title`의 `order` 및 `size` 속성과 `dmc.Text`를 통해 텍스트 계층을 명확히 하고, `dmc.Button`의 `variant` 및 `color`, `dmc.Card`의 `shadow`로 컴포넌트의 중요도를 시각적으로 표현합니다.
    - **반응형 그리드 시스템 사용:** `dmc.Grid` 및 `dmc.GridCol`의 반응형 `span` 속성을 적극 활용하여 다양한 화면 크기에서 콘텐츠가 잘 적응하도록 합니다.
    - **일관된 간격 관리:** `dmc.Grid`의 `gutter`, `dmc.SimpleGrid`의 `spacing`, `dmc.Stack`, `dmc.Group`, `dmc.Flex`의 `gap` 등 모든 레이아웃 컴포넌트에서 `theme.spacing` 토큰을 일관되게 사용하여 여백 및 근접성 원칙을 적용합니다.
    - **`dmc.AppShell`을 통한 구조화:** 애플리케이션의 전반적인 구조를 위해 `dmc.AppShell`을 사용하여 일관된 헤더, 내비게이션, 푸터 등을 설정하고 반응형 중단점을 구성합니다.

3. **데이터 표시 및 피드백의 효율성:**

    - **접근성 있는 이미지:** `dmc.Image` 사용 시 항상 설명적인 `alt` 속성을 제공하고, `withPlaceholder`를 사용하여 이미지 로드 실패 시 우아한 폴백을 제공합니다.
    - **적절한 테이블 사용:** 작은 데이터 세트에는 `dmc.Table`을, 큰 데이터 세트나 고급 기능이 필요할 때는 `dash-ag-grid`를 사용하는 것이 좋습니다.
    - **명확한 피드백:** `dmc.Alert`를 사용하여 사용자에게 중요한 메시지를 전달하고, `dmc.Loader`로 로딩 상태를 시각적으로 알립니다.

4. **데이터 시각화의 명확성:**

    - **올바른 차트 유형 선택:** 데이터와 전달하려는 통찰력에 맞는 차트 유형을 선택합니다.
    - **명확하고 간결한 차트:** 차트를 깔끔하게 유지하고 불필요한 "차트 쓰레기"를 피하며, 레이블과 범례는 명확하고 가독성 있게 만듭니다.
    - **테마 색상 사용:** 차트 시리즈 색상에 `theme.colors`를 우선적으로 사용하여 일관성을 유지합니다.
    - **접근성 및 설명:** 차트의 모든 텍스트 요소에 충분한 대비와 가독성을 보장하고, `dmc.Title` 또는 `dmc.Text`를 통해 차트 내용을 설명합니다.

5. **타이포그래피의 전략적 구현:**

    - **의미론적 제목 사용:** `dmc.Title`을 사용하여 `h1`~`h6` 태그의 논리적 문서 구조를 유지합니다.
    - **본문 텍스트 가독성:** `dmc.Text`를 사용하여 본문 텍스트의 `size` 및 `lineHeight`가 가독성에 최적화되도록 합니다. `c="dimmed"`와 같은 색상으로 보조 정보를 효과적으로 표현합니다.
    - **코드 표시:** `dmc.Code`를 인라인 또는 블록 형태로 사용하여 코드 스니펫을 명확하게 표시합니다.

이러한 모범 사례들을 따르면 Dash Mantine Components를 활용하여 일관되고, 접근 가능하며, 사용자 친화적인 애플리케이션을 효율적으로 개발할 수 있습니다.

마크다운 문서를 기준으로 프로젝트 설계 구조 제안

이 마크다운 문서는 Dash Mantine Components (DMC) 기반 Dash 애플리케이션의 UI/UX 설계에 대한 포괄적인 가이드라인을 제공합니다. 이 가이드라인을 바탕으로, OOP (객체 지향 프로그래밍) 및 SOLID 원칙을 준수하고 유지보수/확장성이 좋은 아키텍처를 지향하는 파이썬 개발 환경에 맞춰 프로젝트 설계 구조를 제안합니다.

제안하는 프로젝트 구조:

```
.
├── app.py                      # Dash 앱 인스턴스 및 최상위 레이아웃 정의
├── index.py                    # 다중 페이지 앱의 경우 콜백 및 라우팅
├── config/                     # 애플리케이션 전반의 설정 (환경 변수, 상수 등)
│   ├── __init__.py
│   └── settings.py
├── assets/                     # 정적 파일 (CSS, JS, 이미지, 로고 등)
│   ├── css/
│   ├── js/
│   └── images/
├── components/                 # 재사용 가능한 UI 컴포넌트 모듈 (객체 지향적으로 구
성)
│   ├── __init__.py
│   ├── layout/                 # AppShell, Grid, Stack 등을 캡슐화한 레이아웃 컴
포넌트
│   │   ├── __init__.py
│   │   ├── app_shell.py
│   │   └── responsive_grid.py
│   ├── data_display/           # Table, Image, Chart 등을 캡슐화한 컴포넌트
│   │   ├── __init__.py
│   │   ├── custom_table.py
│   │   └── dynamic_chart.py
│   ├── feedback/               # Alert, Loader 등을 캡슐화한 컴포넌트
│   │   ├── __init__.py
│   │   └── notification_banner.py
│   └── common/                 # 범용적으로 사용될 UI 요소 (Button, Text 등)
│       └── custom_button.py
├── pages/                      # 다중 페이지 앱의 각 페이지 정의
│   ├── __init__.py
│   ├── home/
│   │   ├── __init__.py
│   │   ├── layout.py           # 페이지별 레이아웃 정의
│   │   └── callbacks.py        # 페이지별 콜백 정의
│   ├── dashboard/
│   │   ├── __init__.py
│   │   ├── layout.py
│   │   └── callbacks.py
│   └── ...
├── themes/                     # Mantine 테마 정의
│   ├── __init__.py
│   └── main_theme.py           # dmc.MantineProvider의 theme 속성 정의
├── utils/                      # 유틸리티 함수 (데이터 처리, 헬퍼 함수 등)
│   ├── __init__.py
│   ├── data_utils.py
│   └── validation_utils.py
├── services/                   # 데이터 서비스 또는 API 호출 로직 (비즈니스 로직 분
```

```
리)
│   ├── __init__.py
│   └── data_service.py
├── tests/                      # 테스트 코드 (TDD 준수)
│   ├── __init__.py
│   ├── components/
│   │   └── test_custom_button.py
│   ├── pages/
│   │   └── test_home_layout.py
│   └── services/
│       └── test_data_service.py
├── Dockerfile                  # (선택 사항) 배포를 위한 Docker 설정
├── requirements.txt            # 프로젝트 의존성
├── .env                        # 환경 변수 (예: 개발/운영 모드)
├── .pylintrc                   # Pylint 설정
├── pyproject.toml              # Mypy 설정 (PEP 518) 또는 기타 프로젝트 설정
└── README.md
```

설계 원칙 및 이점:

1. **객체 지향 프로그래밍 (OOP) 및 SOLID 원칙**:
    - **단일 책임 원칙 (SRP)**: 각 모듈 또는 파일은 단일 기능(예: `app_shell.py`는 `AppShell` 레이아웃만 정의)을 담당합니다.
    - **개방-폐쇄 원칙 (OCP)**: `components/` 디렉토리의 UI 컴포넌트들은 상속 또는 구성(Composition)을 통해 확장될 수 있지만, 기존 코드는 수정되지 않도록 설계합니다. 예를 들어, `CustomButton` 클래스는 `dmc.Button`을 상속하거나 래핑하여 특정 스타일이나 동작을 추가할 수 있습니다.
    - **리스코프 치환 원칙 (LSP)**: `components` 내의 사용자 정의 컴포넌트들은 `dmc` 컴포넌트와 동일한 방식으로 사용될 수 있도록 인터페이스를 유지합니다.
    - **인터페이스 분리 원칙 (ISP)**: 필요한 경우, 대규모 인터페이스보다는 특정 목적을 위한 작은 인터페이스를 구성합니다. (파이썬에서는 명시적인 인터페이스보다 Duck Typing에 가깝게 적용)
    - **의존성 역전 원칙 (DIP)**: `app.py`나 `pages`의 콜백 등 상위 모듈이 하위 수준의 구현(예: `services/data_service.py`)에 직접 의존하기보다는 추상화에 의존하도록 합니다. 데이터 서비스 로직을 별도의 서비스 레이어로 분리하여 UI 로직으로부터 독립적으로 테스트하고 교체할 수 있도록 합니다.
2. **모듈화 및 재사용성**:
    - `components/` 디렉토리의 컴포넌트들은 여러 페이지에서 재사용될 수 있도록 일반적인 UI 패턴을 캡슐화합니다. 예를 들어, `responsive_grid.py`는 `dmc.Grid`의 반응형 동작을 미리 정의하여 다른 페이지에서 일관된 레이아웃을 빠르게 구성할 수 있습니다.
    - `themes/main_theme.py`에 전역 테마를 중앙 집중화하여 애플리케이션 전체의 스타일 일관성을 쉽게 관리하고 변경할 수 있습니다.
3. **유지보수성 및 확장성**:
    - 명확한 디렉토리 구조는 코드 베이스를 이해하고 탐색하기 쉽게 만듭니다.
    - 페이지별로 `layout.py`와 `callbacks.py`를 분리하여 대규모 페이지의 콜백 로직을 관리하기 용이하게 합니다.

- `services/` 레이어는 비즈니스 로직과 데이터 접근을 분리하여 UI 변경이 핵심 로직에 영향을 미치지 않도록 합니다.

4. **TDD (테스트 주도 개발) 지원:**
   - `tests/` 디렉토리는 단위 테스트, 통합 테스트, UI 컴포넌트 테스트를 위한 공간을 제공합니다. 개발 초기부터 테스트 코드를 작성하도록 장려하여 코드 품질과 안정성을 높입니다.
   - `pylint` 및 `mypy` 설정 파일을 포함하여 코드 일관성과 타입 안전성을 강제합니다.

이러한 구조는 문서를 통해 강조된 DMC의 테마, 레이아웃, 컴포넌트 사용 원칙을 Python 프로젝트 구조에 반영하여, 깔끔하고 유용하며 확장 가능한 Dash 애플리케이션 개발을 지원합니다.

Python 코드와 연결된 UI 설계 가이드 추출

문서에서 AI에게 직접적으로 Python 코드 패턴 및 DMC 속성 구성과 연결하여 UI를 설계하도록 지시하는 가이드라인을 추출했습니다. 이는 AI가 추상적인 UI/UX 원칙을 실제 코드로 전환하는 데 필요한 "실행 가능한 지침"입니다.

1. **핵심 테마 설정 (`dmc.MantineProvider` `theme` 객체 사용)**
   - **브랜드 색상 정의:** 애플리케이션의 브랜드 아이덴티티에 따라 `primaryColor`를 정의하고, `theme.colors` 딕셔너리에 10가지 음영을 가진 사용자 정의 색상 팔레트를 정의합니다.

```
# 예시: main_theme.py
import dash_mantine_components as dmc

APP_THEME = {
    "primaryColor": "myBrandBlue",
    "colors": {
        "myBrandBlue": [
            "#E7F5FF", "#D0EBFF", "#A5D8FF", "#74C0FC",
"#4DA6FF",
            "#2C7BEF", "#1D62EA", "#1864AB", "#14457F", "#10325D"
        ],
        # ... 다른 브랜드 색상 정의
    },
    # ...
}
# dmc.MantineProvider(theme=APP_THEME, ...)
```

   - **기본 음영 최적화:** `primaryShade`를 조정하여 밝은/어두운 테마 모두에서 기본 색상 배경의 텍스트 및 아이콘에 충분한 대비를 보장합니다 (접근성).

```
APP_THEME = {
    # ...
    "primaryShade": {"light": 6, "dark": 8},
    # ...
}
```

- **전역 글꼴 및 제목 설정:** `theme.fontFamily`로 기본 글꼴을 설정하고, `theme.headings.sizes`를 구성하여 `h1`부터 `h6`까지 시각적 계층 구조를 명확히 합니다.

```
APP_THEME = {
    # ...
    "fontFamily": "Inter, sans-serif",
    "headings": {
        "fontFamily": "Roboto, sans-serif",
        "fontWeight": 700,
        "sizes": {
            "h1": {"fontSize": "2.5rem", "lineHeight": 1.3},
            "h2": {"fontSize": "2rem", "lineHeight": 1.35,
"fontWeight": 600},
            "h3": {"fontSize": "1.75rem", "lineHeight": 1.4},
            # ...
        }
    },
    "fontSizes": {"md": "1rem"}, # 본문 텍스트 기본 16px
    "lineHeights": {"md": 1.55}, # 본문 텍스트 기본 줄 높이
    # ...
}
```

- **일관된 간격 및 곡선:** `theme.spacing`을 4px 또는 8px의 배수를 기반으로 정의하고, `defaultRadius`를 설정하여 전반적인 일관성을 유지합니다.

```
APP_THEME = {
    # ...
    "spacing": {
        "xs": "0.25rem", "sm": "0.5rem", "md": "1rem",
        "lg": "1.5rem", "xl": "2rem"
    },
    "radius": {"sm": "0.25rem", "md": "0.5rem"},
    "defaultRadius": "md",
    "shadows": { # 부드럽고 일관된 그림자 시스템 정의
        "sm": "0 1px 3px rgba(0,0,0,0.05), 0 1px 2px
rgba(0,0,0,0.1)",
    },
    # ...
}
```

- **컴포넌트 기본값 설정:** `theme.components`를 사용하여 자주 사용되는 컴포넌트의 `defaultProps`를 정의하여 반복적인 스타일링을 피하고 일관성을 보장합니다.

```
APP_THEME = {
    # ...
    "components": {
        "Button": {
            "defaultProps": {"variant": "filled", "color":
```

```
"primary", "radius": "md"}
        },
        "Card": {
            "defaultProps": {"shadow": "sm", "withBorder": True,
"padding": "lg"}
        },
        "TextInput": {
            "styles": lambda theme, props: {
                "input": {"borderColor": theme["colors"]["gray"]
[4]}
            }
        }
    }
}
```

2. **시각적 계층 구조를 위한 컴포넌트 사용**:
   - **제목**: 모든 페이지 및 섹션 제목에 `dmc.Title`을 사용하고 `order` 속성을 논리적 문서 구조에 맞게 설정합니다. `size` 및 `fw`로 시각적 중요도를 부여합니다.

```
dmc.Title("애플리케이션 대시보드", order=1, fz="2.5rem")
dmc.Title("데이터 요약", order=2, fz="2rem", fw=600)
```

   - **본문 텍스트**: `dmc.Text`를 사용하여 본문 내용을 렌더링하고 `c="dimmed"`를 사용하여 보조 정보를 강조합니다.

```
dmc.Text("이것은 대시보드의 주요 정보입니다.", size="md", c="dark.7")
dmc.Text("추가 세부 정보는 아래에서 확인하십시오.", size="sm",
c="dimmed")
```

   - **버튼 계층**: 주요 Call-to-Action에는 `variant="filled"` 및 `color="primary"`를 사용하고, 보조 작업에는 `variant="light"` 또는 `outline`을 사용합니다.

```
dmc.Button("데이터 저장", color="primary", variant="filled")
dmc.Button("취소", variant="outline")
```

3. **반응형 레이아웃 구현 (그리드 및 Flexbox)**:
   - **`dmc.GridCol`의 반응형 `span`**: 주요 콘텐츠 영역을 `dmc.Grid`로 구조화하고, `dmc.GridCol`의 `span` 속성에 반응형 딕셔너리를 사용하여 화면 크기에 따라 열 너비를 조정합니다.

```
dmc.Grid(gutter="md", children=[
    dmc.GridCol(span={"base": 12, "sm": 6, "lg": 4}, children="카
드 1"),
    dmc.GridCol(span={"base": 12, "sm": 6, "lg": 4}, children="카
```

```
드 2"),
    dmc.GridCol(span={"base": 12, "sm": 12, "lg": 4}, children="카
드 3"),
])
```

- **`dmc.SimpleGrid` 사용**: 균등한 크기의 항목이 필요한 레이아웃에 `dmc.SimpleGrid`를 사용하고 `cols` 속성으로 반응형 열 수를 제어합니다.

```
dmc.SimpleGrid(cols={"base": 1, "sm": 2, "md": 3}, spacing="lg",
children=[
    dmc.Card("아이템 A"), dmc.Card("아이템 B"), dmc.Card("아이템 C")
])
```

- **`dmc.Stack`, `dmc.Group`, `dmc.Flex`의 `gap`**: 수직 또는 수평 정렬 시 `gap` 속성에 `theme.spacing` 키를 사용하여 일관된 간격을 유지합니다.

```
dmc.Stack(gap="sm", children=[
    dmc.TextInput(label="이름"), dmc.TextInput(label="이메일")
])
dmc.Group(gap="md", justify="flex-end", children=[
    dmc.Button("저장"), dmc.Button("삭제")
])
```

4. **애플리케이션 셸 구조화 (`dmc.AppShell`)**:

- 표준 애플리케이션 인터페이스에는 `dmc.AppShell`을 사용하고, `header`, `navbar` 속성에 높이/너비 및 `breakpoint` 설정을 구성하여 반응형 동작을 보장합니다.

```
dmc.AppShell(
    header={"height": 60, "withBorder": True},
    navbar={"width": 250, "breakpoint": "sm", "collapsed":
{"mobile": True, "desktop": False}},
    children=[
        dmc.AppShellHeader(children=dmc.Text("애플리케이션 헤더")),
        dmc.AppShellNavbar(children=dmc.Text("탐색 메뉴")),
        dmc.AppShellMain(children=dmc.Text("메인 콘텐츠 영역"))
    ]
)
```

5. **데이터 시각화 및 피드백 (`dmc.Chart` 및 `dmc.Alert`)**:

- **차트 데이터 구조화**: 차트 컴포넌트 사용 시 `data`, `dataKey`, `series` 속성이 올바르게 구조화되었는지 확인합니다.
- **정보성 알림**: `dmc.Alert`를 사용하여 양식 유효성 검사 오류, 성공 메시지 등 사용자에게 중요한 피드백을 제공하고, `color` 및 `icon` 속성을 사용하여 시각적 신호를 강화합니다.

```
# 성공 메시지
dmc.Alert(
    "데이터가 성공적으로 저장되었습니다.",
    title="성공",
    color="green",
    icon=DashIconify(icon="tabler:check-circle"),
    duration=5000,
    withCloseButton=True
)
# 오류 메시지
dmc.Alert(
    "입력 필드를 확인하십시오.",
    title="유효성 검사 오류",
    color="red",
    icon=DashIconify(icon="tabler:alert-circle"),
    withCloseButton=True
)
```

이러한 지침은 AI가 Dash Mantine Components를 활용하여 OOP 및 SOLID 원칙을 기반으로 하는 Pythonic한 코드를 통해 깔끔하고 유용한 UI를 효과적으로 생성하도록 돕는 구체적인 설계 프레임워크를 제공합니다.