La Guía Definitiva del Desarrollador para Tremor: Componentes, Configuración y Personalización

Parte I: Fundamentos y Configuración

Esta parte fundamental del informe abordará los conceptos centrales del ecosistema Tremor. Aclarará una ambigüedad crítica identificada durante la investigación: la existencia de dos bibliotecas Tremor distintas, "Tremor Raw" y la versión con temas @tremor/react. Esta distinción impacta fundamentalmente el enfoque de un desarrollador hacia la instalación, el uso y la personalización.

Sección 1: Introducción al Ecosistema Tremor

Filosofía Central

La misión de Tremor es proporcionar a los desarrolladores componentes de React listos para copiar y pegar, permitiendo la construcción rápida y eficiente de aplicaciones web modernas, con un enfoque particular en gráficos y dashboards.[1] El énfasis se pone en ofrecer primitivas de interfaz de usuario (UI) accesibles y listas para producción, diseñadas por científicos de datos e ingenieros de software con una especial atención al diseño.[2]

Tecnologías Subyacentes

La biblioteca se construye sobre una base tecnológica robusta y moderna que incluye React, Tailwind CSS y Radix UI.[1] Esta combinación promete componentes de alta calidad, accesibles y estilísticamente flexibles. La elección de estas tecnologías subyacentes es un punto clave de venta, ya que aprovecha herramientas líderes en la industria para el desarrollo de interfaces de usuario:

- **React:** Proporciona el marco declarativo y basado en componentes para construir la UI.
- **Tailwind CSS:** Ofrece un framework de CSS de bajo nivel y basado en utilidades que permite una personalización de estilos profunda y sistemática.
- **Radix UI:** Suministra un conjunto de primitivas de UI sin estilo, accesibles y de bajo nivel que sirven como la base funcional para muchos de los componentes más complejos de Tremor, garantizando que la accesibilidad sea una característica fundamental y no una ocurrencia tardía.[2]

La Dicotomía de "Dos Tremors"

Un análisis profundo de la documentación oficial revela la existencia de dos paradigmas distintos dentro del ecosistema Tremor, una distinción crucial que todo desarrollador debe comprender antes de iniciar un proyecto.

- **Tremor Raw:** La documentación principal en tremor.so/docs describe "Tremor Raw", una colección de componentes funcionales pero sin estilos predefinidos.[5] La personalización en este paradigma se logra aplicando manualmente clases de utilidad de Tailwind CSS directamente a los componentes. Este enfoque ofrece el máximo control y flexibilidad, ideal para proyectos con un sistema de diseño altamente personalizado donde la adhesión a un tema preexistente no es deseable.
- **@tremor/react (Versión con Temas):** La investigación suplementaria, incluyendo la documentación en npm.tremor.so y varias guías de integración, apunta a una biblioteca más antigua y "llave en mano" distribuida bajo el paquete de npm @tremor/react.[6] Esta versión viene con un sofisticado sistema de temas incorporado que se configura a través del archivo tailwind.config.js. Está diseñada para una implementación rápida, proporcionando un aspecto cohesivo y profesional desde el primer momento.

Esta división parece reflejar una evolución estratégica en la filosofía de la biblioteca. La versión @tremor/react ofrece una solución rápida y consistente, mientras que Tremor Raw se alinea con la tendencia moderna de componentes "headless" o sin estilo, otorgando a los desarrolladores un control granular total sobre la apariencia. Para un desarrollador, la elección es fundamental: optar por la biblioteca con temas para velocidad y consistencia, o elegir la biblioteca "raw" para un control de diseño absoluto desde cero. Este informe

documentará ambos sistemas para proporcionar una cobertura exhaustiva que satisfaga cualquier necesidad de proyecto.

Visión General del Ecosistema

Más allá de la biblioteca de componentes principal, el ecosistema de Tremor incluye varios recursos diseñados para acelerar los flujos de trabajo de desarrollo y diseño:

- **Tremor Blocks:** Son fragmentos de código listos para producción que combinan múltiples componentes para formar secciones de UI comunes, como tarjetas de KPI o filtros complejos. Permiten a los desarrolladores copiar y pegar bloques enteros en sus aplicaciones, acelerando significativamente la construcción de interfaces.[5]
- **Templates:** Tremor ofrece plantillas de aplicaciones completas construidas con Next.js y React, que sirven como puntos de partida para proyectos complejos como dashboards analíticos, sitios de marketing SaaS y páginas de informes.[2]
- **Figma UI Kit:** Para facilitar la colaboración entre diseñadores y desarrolladores, Tremor proporciona un kit de UI para Figma. Este recurso contiene todos los componentes y estilos, permitiendo a los equipos de diseño crear maquetas que se corresponden exactamente con los componentes disponibles en el código, agilizando el proceso de transferencia del diseño al desarrollo.[5]

Sección 2: Instalación y Configuración del Proyecto

Esta sección detalla el proceso de instalación y configuración, centrándose en la versión con temas (@tremor/react) dentro de un proyecto Next.js, ya que representa el caso de uso más documentado y completo.[6]

Prerrequisitos

Es fundamental asegurarse de que el entorno de desarrollo cumpla con los requisitos de versión específicos para la biblioteca Tremor elegida.

- **Para Tremor Raw:**
  - React: v18.2.0 o superior

    - Tailwind CSS: v4.0 o superior [5]
- **Para @tremor/react (Versión con Temas):**
    - React: v18.2.0 o superior
    - Tailwind CSS: v3.4 o superior
    - Headless UI: v2.2.0
    - @tailwindcss/forms: v0.5.9
    - Remix Icon: v4.5.0 [6]

Guía Paso a Paso para Next.js (con @tremor/react)

1. **Crear un nuevo proyecto Next.js:** Inicie un nuevo proyecto utilizando el CLI de Next.js. Se recomienda aceptar las opciones predeterminadas, incluyendo el uso de Tailwind CSS, el directorio src/ y el App Router.
   Bash
   `npx create-next-app@latest my-tremor-dashboard --ts`

2. **Instalar @tremor/react:** Navegue al directorio del proyecto e instale el paquete principal de Tremor.
   Bash
   `npm install @tremor/react`

3. **Instalar Dependencias Adicionales:** Instale las dependencias requeridas para el correcto funcionamiento de los componentes de Tremor.
   Bash
   `npm install @headlessui/react @remixicon/react`
   `npm install -D @tailwindcss/forms`

4. **Verificar Versiones:** Asegúrese de que la versión de Tailwind CSS en su package.json sea al menos 3.4. Si está trabajando en un proyecto existente, actualícela si es necesario.[6]

Configuración de tailwind.config.js

La configuración correcta de tailwind.config.js es el paso más crítico para que Tremor funcione correctamente. Este archivo no solo habilita los estilos de la biblioteca, sino que también actúa como el centro de control para la personalización del tema.

El Array content: Un Paso Crítico

El motor Just-In-Time (JIT) de Tailwind CSS funciona escaneando sus archivos en busca de nombres de clases para generar solo el CSS necesario. Para que este proceso funcione con una biblioteca de terceros como Tremor, es imperativo que le indique a Tailwind dónde encontrar las clases utilizadas por los componentes de Tremor.

Esto se logra añadiendo la ruta al módulo de Tremor dentro del array content en su archivo tailwind.config.js.

JavaScript

```
// tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content:,
  theme: {
    extend: {},
  },
  plugins:,
}
```

La omisión de esta línea es una de las causas más comunes de problemas durante la instalación. Los desarrolladores pueden encontrarse con que los componentes de Tremor se renderizan sin ningún estilo, o solo se estilizan parcialmente si las mismas clases de Tailwind se utilizan casualmente en otra parte de la aplicación.[9] Este comportamiento se debe a que el compilador JIT no "ve" las clases dentro de la carpeta node_modules y, por lo tanto, no las incluye en la hoja de estilos final. Incluir la ruta node_modules/@tremor/**/*.{js,ts,jsx,tsx} resuelve este problema de raíz, garantizando que todos los estilos necesarios se generen correctamente.[6]

Configuración de globals.css

El archivo CSS global es donde se inyectan las directivas base de Tailwind. Asegúrese de que

su archivo globals.css (o su equivalente) contenga las tres directivas principales de Tailwind en la parte superior.

CSS

```
/* globals.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Estas directivas son responsables de inyectar los estilos base de Tailwind, las clases de componentes y las clases de utilidad, respectivamente.[12] Adicionalmente, la documentación de Tremor recomienda aplicar suavizado de fuentes (antialiasing) para una mejor legibilidad del texto en los dashboards, lo cual se puede añadir en el archivo CSS global o directamente en la etiqueta <html>.[6]

Sección 3: Dominando la Personalización y el Theming

Tremor ofrece dos modelos de personalización distintos, alineados con sus dos distribuciones de biblioteca. La elección entre estos modelos dependerá de las necesidades específicas del proyecto en cuanto a control de diseño y velocidad de desarrollo.

El Enfoque de Tremor Raw: Personalización Granular

Con Tremor Raw, la personalización es directa y se basa en el conocimiento de Tailwind CSS. Dado que los componentes vienen sin estilo, el desarrollador tiene la tarea y la libertad de aplicar cualquier clase de utilidad de Tailwind para lograr la apariencia deseada.

Por ejemplo, para personalizar un componente Card de Tremor Raw, se aplicarían clases directamente en el JSX:

JavaScript

```
import { Card } from "@/components/tremor-raw/Card";

function CustomCard() {
  return (
    <Card className="p-6 bg-white dark:bg-gray-900 border border-blue-200 dark:border-blue-800
rounded-lg shadow-md hover:shadow-xl transition-shadow">
      <h3 className="text-lg font-semibold text-gray-900 dark:text-white">Título Personalizado</h3>
      <p className="mt-2 text-sm text-gray-600 dark:text-gray-400">
        Este es el contenido de una tarjeta personalizada con clases de Tailwind.
      </p>
    </Card>
  );
}
```

Este método proporciona un control total, pero requiere un esfuerzo manual para mantener la consistencia visual en toda la aplicación.

El Motor de Theming de @tremor/react: Personalización Semántica

La versión @tremor/react introduce un potente motor de theming que se gestiona centralmente desde el archivo tailwind.config.js. Este sistema está diseñado para una personalización "absoluta" y sistemática, permitiendo cambios de diseño globales con modificaciones mínimas en el código.

La personalización se realiza extendiendo el objeto theme de Tailwind con una estructura de configuración específica de Tremor. A continuación se presenta un análisis detallado de esta configuración, basada en la documentación oficial.[6]

Desglosando la Configuración del Tema

El siguiente es un ejemplo completo de la sección theme en tailwind.config.js para habilitar el theming de Tremor:

JavaScript

```ts
// tailwind.config.ts
import type { Config } from "tailwindcss";
import colors from "tailwindcss/colors";

const config: Config = {
  content: [
    "./src/**/*.{js,ts,jsx,tsx}",
    "./node_modules/@tremor/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    transparent: "transparent",
    current: "currentColor",
    extend: {
      colors: {
        // Modo claro
        tremor: {
          brand: {
            faint: colors.blue,
            muted: colors.blue,
            subtle: colors.blue,
            DEFAULT: colors.blue,
            emphasis: colors.blue,
            inverted: colors.white,
          },
          background: {
            muted: colors.gray,
            subtle: colors.gray,
            DEFAULT: colors.white,
            emphasis: colors.gray,
          },
          border: {
            DEFAULT: colors.gray,
          },
          ring: {
            DEFAULT: colors.gray,
          },
          content: {
            subtle: colors.gray,
            DEFAULT: colors.gray,
            emphasis: colors.gray,
            strong: colors.gray,
```

```
          inverted: colors.white,
        },
      },
      // Modo oscuro
      "dark-tremor": {
        brand: {
          faint: "#0B1229",
          muted: colors.blue,
          subtle: colors.blue,
          DEFAULT: colors.blue,
          emphasis: colors.blue,
          inverted: colors.blue,
        },
        background: {
          muted: "#131A2B",
          subtle: colors.gray,
          DEFAULT: colors.gray,
          emphasis: colors.gray,
        },
        border: {
          DEFAULT: colors.gray,
        },
        ring: {
          DEFAULT: colors.gray,
        },
        content: {
          subtle: colors.gray,
          DEFAULT: colors.gray,
          emphasis: colors.gray,
          strong: colors.gray,
          inverted: colors.gray,
        },
      },
    },
    boxShadow: {
      "tremor-input": "0 1px 2px 0 rgb(0 0 0 / 0.05)",
      "tremor-card": "0 1px 3px 0 rgb(0 0 0 / 0.1), 0 1px 2px -1px rgb(0 0 0 / 0.1)",
      "tremor-dropdown": "0 4px 6px -1px rgb(0 0 0 / 0.1), 0 2px 4px -2px rgb(0 0 0 / 0.1)",
      "dark-tremor-input": "0 1px 2px 0 rgb(0 0 0 / 0.05)",
      "dark-tremor-card": "0 1px 3px 0 rgb(0 0 0 / 0.1), 0 1px 2px -1px rgb(0 0 0 / 0.1)",
      "dark-tremor-dropdown": "0 4px 6px -1px rgb(0 0 0 / 0.1), 0 2px 4px -2px rgb(0 0 0 / 0.1)",
    },
    borderRadius: {
```

```
        "tremor-small": "0.375rem",
        "tremor-default": "0.5rem",
        "tremor-full": "9999px",
      },
      fontSize: {
        "tremor-label": ["0.75rem", { lineHeight: "1rem" }],
        "tremor-default": ["0.875rem", { lineHeight: "1.25rem" }],
        "tremor-title": ["1.125rem", { lineHeight: "1.75rem" }],
        "tremor-metric": ["1.875rem", { lineHeight: "2.25rem" }],
      },
    },
  },
  plugins: [require("@tailwindcss/forms")],
};
export default config;
```

La Abstracción del Theming Semántico

El diseño del objeto theme de Tremor es intencionalmente abstracto. En lugar de definir colores directamente (por ejemplo, primary: 'blue'), utiliza una estructura semántica que describe el *propósito* del color. Las categorías principales son brand, background, border, ring y content. Cada una de estas categorías se subdivide en niveles de intensidad o propósito, como faint, muted, subtle, DEFAULT, emphasis, strong e inverted.[6]

Este enfoque semántico es la clave para una personalización potente y mantenible. Cuando un desarrollador modifica el valor de tremor.brand.DEFAULT, no está simplemente cambiando un color; está cambiando el "color de marca principal". Este cambio se propagará automáticamente a todos los componentes que utilizan ese concepto semántico, como los botones primarios, los elementos activos en un gráfico o los bordes de los inputs enfocados. Esto asegura una consistencia visual en toda la aplicación y reduce drásticamente el esfuerzo necesario para realizar cambios de marca o de tema.

Personalización más allá de los Colores

El sistema de theming se extiende más allá de los colores, permitiendo un control sobre otros aspectos fundamentales del diseño:

- **Sombras (boxShadow):** Se pueden definir sombras personalizadas para elementos

específicos como input, card y dropdown, con variantes para los modos claro y oscuro.[6]
- **Radio de Borde (borderRadius):** Permite establecer radios de borde consistentes para diferentes tamaños de componentes (small, default, full).[7]
- **Tamaños de Fuente (fontSize):** Define una escala tipográfica semántica para elementos como label, title y metric, asegurando una jerarquía visual clara y consistente.[6]

Salvaguardando Estilos con safelist

Algunos componentes de Tremor, como Badge o los gráficos, aceptan una prop color que aplica dinámicamente clases de color (por ejemplo, bg-red-500, text-emerald-700). En un entorno de producción, el proceso de purga de Tailwind CSS podría eliminar estas clases si no las encuentra utilizadas estáticamente en el código.

Para evitar esto, se puede utilizar la opción safelist en tailwind.config.js. Esta configuración le indica a Tailwind que nunca elimine ciertas clases, garantizando que los colores dinámicos de los componentes de Tremor siempre funcionen.[7]

JavaScript

```
// tailwind.config.js
module.exports = {
  //...
  safelist: [
    {
      pattern:
/^(bg-(?:slate|gray|zinc|neutral|stone|red|orange|amber|yellow|lime|green|emerald|teal|cyan|sky|blue|indigo|violet|purple|fuchsia|pink|rose)-(?:50|100|200|300|400|500|600|700|800|900|950))$/,
      variants: ["hover", "ui-selected"],
    },
    //... patrones similares para text-, border-, ring-, stroke-, fill-
  ],
  //...
};
```

Parte II: La Referencia Completa de Componentes

Esta parte sirve como una enciclopedia exhaustiva de cada componente disponible en Tremor, organizada por su categoría funcional. Cada subsección de componente sigue un formato estandarizado para facilitar la consulta y referencia, incluyendo su propósito, una tabla detallada de su API de propiedades (props) y ejemplos de código anotados que demuestran su uso desde lo básico hasta lo avanzado.

Sección 4: Componentes de Visualización

Los componentes de visualización son el núcleo de Tremor, diseñados para transformar datos en gráficos y representaciones visuales claras e interactivas.

Area Chart (Gráfico de Área)

- **Propósito y Casos de Uso:** El AreaChart se utiliza para mostrar datos cuantitativos a lo largo de un eje continuo, generalmente el tiempo. Es ideal para visualizar tendencias, volúmenes y comparar la contribución de diferentes categorías a un total a lo largo del tiempo. Las áreas sombreadas bajo las líneas ayudan a enfatizar la magnitud de los valores.[13]
- **Tabla de Referencia de API:**

Prop	Tipo	Descripción	Valor por Defecto
data	Record<string, any>	Array de objetos de datos para renderizar el gráfico. **Requerido.**	undefined
index	string	La clave en los objetos de datos que se usará para el eje X. **Requerido.**	undefined

categories	string	Array de claves que representan las series de datos a trazar. **Requerido.**	undefined
type	'default' | 'stacked' | 'percent'	Define el tipo de gráfico de área.	'default'
colors	AvailableChartColorsKeys	Array de colores para las categorías.	Paleta por defecto
valueFormatter	(value: number) => string	Función para formatear los valores mostrados en el eje Y y en los tooltips.	(val) => val
onValueChange	(value: EventProps) => void	Callback que se activa al hacer clic en un punto de datos, haciendo el gráfico interactivo.	undefined
showLegend	boolean	Controla la visibilidad de la leyenda.	true
showTooltip	boolean	Controla la visibilidad del tooltip.	true
showGridLines	boolean	Controla la visibilidad de las líneas de la cuadrícula.	true
xAxisLabel	string	Etiqueta para el eje X.	undefined

yAxisLabel	string	Etiqueta para el eje Y.	undefined
startEndOnly	boolean	Muestra solo la primera y última etiqueta en el eje X.	false
fill	'solid' | 'gradient' | 'none'	Define el estilo de relleno del área.	'gradient'

- **Ejemplos de Código Anotados:**
  - **Uso Básico:**

```
JavaScript
const chartdata =;

<AreaChart
 className="h-80"
 data={chartdata}
 index="date"
 categories={}
 valueFormatter={(number) => `$${Intl.NumberFormat("us").format(number).toString()}`}
 onValueChange={(v) => console.log(v)}
/>
```

    Este ejemplo renderiza un gráfico de área comparando dos categorías, formatea los valores como moneda y registra los eventos de clic en la consola.[13]
  - **Gráfico Apilado con Etiquetas de Eje:**

```
JavaScript
<AreaChart
 type="stacked"
 data={chartdata}
 index="date"
 categories={}
 xAxisLabel="Month"
 yAxisLabel="Spend Category"
 fill="solid"
/>
```

    Aquí, el tipo se establece en stacked para apilar los valores, se añaden etiquetas a los ejes y el relleno se cambia a un color sólido.[13]

Bar Chart (Gráfico de Barras)

- **Propósito y Casos de Uso:** El BarChart es ideal para comparar valores entre diferentes categorías discretas. Utiliza barras verticales u horizontales para representar los datos, facilitando la identificación de los valores más altos y más bajos de un vistazo.[13]
- **Tabla de Referencia de API:** (Similar a AreaChart, con props como data, index, categories, valueFormatter, onValueChange, xAxisLabel, yAxisLabel, startEndOnly).
- **Ejemplos de Código Anotados:**
  - **Gráfico de Barras Agrupado:**

    ```
    JavaScript
    const chartdata =;

    <BarChart
     data={chartdata}
     index="name"
     categories={}
     yAxisWidth={48}
    />
    ```

    Este ejemplo muestra múltiples categorías para cada punto del eje X, creando un gráfico de barras agrupado que es útil para comparaciones detalladas.[13]
  - **Gráfico Minimalista:**

    ```
    JavaScript
    <BarChart
     data={tempData}
     index="hour"
     categories={["temperature"]}
     startEndOnly={true}
     showLegend={false}
     showTooltip={false}
     xAxisLabel="24H Temperature Readout (Zurich)"
    />
    ```

    Configuración para una visualización más limpia, mostrando solo los puntos de inicio y fin en el eje X y ocultando la leyenda y el tooltip.[13]

Combo Chart (Gráfico Combinado)

- **Propósito y Casos de Uso:** El ComboChart permite combinar un gráfico de barras y un gráfico de líneas en una sola visualización. Es extremadamente útil para mostrar datos

con diferentes escalas o unidades, utilizando un eje Y secundario (biaxial).[13]

- **Tabla de Referencia de API:** Incluye props generales (data, index) y objetos de configuración específicos para barSeries y lineSeries, cada uno con sus propias categories, colors, yAxisLabel, etc. La prop clave es enableBiaxial para activar el segundo eje Y.[13]
- **Ejemplo de Código Anotado:**

```JavaScript
<ComboChart
 data={chartdata}
 index="date"
 enableBiaxial={true}
 barSeries={{
  categories:,
  yAxisLabel: "Solar Panels (Bars)",
 }}
 lineSeries={{
  categories: ["Inverters"],
  yAxisLabel: "Inverters (Line)",
  colors: ["amber"],
 }}
/>
```

Este ejemplo traza "SolarPanels" como barras en el eje Y izquierdo y "Inverters" como una línea en el eje Y derecho, permitiendo una comparación directa de dos métricas diferentes.[13]

Donut Chart (Gráfico de Rosquilla / Tarta)

- **Propósito y Casos de Uso:** El DonutChart se utiliza para mostrar la composición de un todo, representando proporciones o porcentajes de diferentes categorías. La variante pie crea un gráfico de tarta tradicional, mientras que la variante donut (por defecto) deja un espacio en el centro que puede usarse para mostrar información adicional, como el valor total.[13]
- **Tabla de Referencia de API:**

Prop	Tipo	Descripción	Valor por Defecto
data	Record<string, any>	Array de datos. **Requerido.**	undefined

category	string	Clave para los nombres de las categorías. **Requerido.**	undefined
value	string	Clave para los valores numéricos. **Requerido.**	undefined
variant	'donut' | 'pie'	El tipo de gráfico circular.	'donut'
label	React.ReactNode	Contenido para mostrar en el centro del gráfico (solo en donut).	undefined
showLabel	boolean	Controla la visibilidad de la etiqueta central.	false
onValueChange	(value: DonutChartEventProps) => void	Callback que hace que los segmentos del gráfico sean interactivos.	undefined

- **Ejemplo de Código Anotado:**

```
JavaScript
const data =;

<DonutChart
 data={data}
 category="name"
 value="amount"
 showLabel={true}
 valueFormatter={(number) => `$${Intl.NumberFormat("us").format(number).toString()}`}
 onValueChange={(v) => console.log(v)}
/>
```

Este ejemplo crea un gráfico de rosquilla, muestra una etiqueta en el centro (probablemente el total), formatea los valores como moneda y permite la interacción del

usuario.[13]

Otros Componentes de Visualización (Resumen)

- **Bar List:** Muestra una lista de barras horizontales, ideal para clasificaciones o rankings. Es simple, compacto y fácil de leer. Se puede ordenar y hacer interactivo.[13]
- **Category Bar:** Una sola barra segmentada que muestra la proporción de diferentes categorías dentro de un total. Útil para representar el progreso hacia un objetivo o la composición de una métrica.[13]
- **Line Chart:** Similar al AreaChart pero sin el relleno, enfocándose puramente en la tendencia y el cambio de los datos a lo largo del tiempo. Conecta puntos de datos con líneas rectas.[13]
- **Progress Bar:** Un indicador lineal que muestra el progreso de una tarea o el estado de una métrica en una escala de 0 a 100. Viene en varias variantes de color (default, success, warning, error).[13]
- **Progress Circle:** Un indicador circular para el progreso. Al igual que la barra de progreso, es ideal para mostrar porcentajes y puede contener contenido en su centro.[13]
- **Spark Chart:** Gráficos en miniatura (SparkAreaChart, SparkLineChart, SparkBarChart) diseñados para ser incrustados en tablas, tarjetas o texto. Proporcionan un contexto visual rápido de la tendencia de los datos sin ocupar mucho espacio.[13]
- **Tracker:** Una serie de bloques de colores utilizados para visualizar el estado a lo largo del tiempo, como el tiempo de actividad de un servicio. Cada bloque puede tener un color y un tooltip para representar un estado específico (por ejemplo, operativo, error, advertencia).[13]

Sección 5: Componentes de Entrada y Formularios

Estos componentes son esenciales para recopilar información del usuario y permitir la interacción dentro de los dashboards y aplicaciones.

Calendar

- **Propósito y Casos de Uso:** Un componente para seleccionar una fecha única o un rango de fechas. Se basa en la API de React Day Picker, ofreciendo una amplia personalización,

incluyendo localización, navegación por año y deshabilitación de fechas específicas.[14]

Checkbox

- **Propósito y Casos de Uso:** Una casilla de verificación estándar que permite al usuario seleccionar una o más opciones. Admite un estado indeterminate, útil en interfaces jerárquicas (por ejemplo, seleccionar algunos, pero no todos, los elementos de una lista).[14]

Date Picker y Date Range Picker

- **Propósito y Casos de Uso:** Componentes de entrada de alto nivel construidos sobre el Calendar. El DatePicker permite seleccionar una sola fecha, mientras que el DateRangePicker permite seleccionar un período. Ambos pueden incluir preajustes (presets) como "Últimos 7 días" o "Mes hasta la fecha" para una selección rápida.[14]

Dropdown Menu

- **Propósito y Casos de Uso:** Presenta una lista de opciones o acciones en un menú contextual que aparece al hacer clic en un disparador. Es altamente versátil y puede contener elementos simples, grupos, CheckboxItem, RadioItem y submenús anidados.[14]

Input

- **Propósito y Casos de Uso:** Un campo de entrada de formulario versátil. Admite varios tipos como text, password (con un botón para mostrar/ocultar), search (con un ícono), number (con controles de paso) y file. También tiene estados visuales para disabled y hasError.[14]

Label

- **Propósito y Casos de Uso:** Proporciona una etiqueta accesible para los elementos de entrada del formulario. Se vincula a un input a través de los atributos htmlFor e id, mejorando la usabilidad y la accesibilidad.[14]

Radio Card Group y Radio Group

- **Propósito y Casos de Uso:** Permiten al usuario seleccionar una única opción de un conjunto. El RadioGroup utiliza botones de radio estándar. El RadioCardGroup ofrece una presentación visualmente más rica, donde cada opción se presenta como una tarjeta seleccionable, ideal para planes de precios o configuraciones visuales.[14]

Select y Select Native

- **Propósito y Casos de Uso:** Ambos componentes presentan una lista desplegable de opciones. Select es un componente personalizado y estilizado que ofrece más flexibilidad de diseño, incluyendo grupos, iconos y estados de error. SelectNative utiliza el elemento <select> nativo del navegador, lo que puede ser beneficioso para el rendimiento y la accesibilidad en ciertos casos, aunque con menos opciones de estilo.[14]

Slider

- **Propósito y Casos de Uso:** Permite al usuario seleccionar un valor o un rango de valores dentro de un mínimo y un máximo definidos, arrastrando un control a lo largo de una pista. Admite orientación vertical y horizontal, pasos definidos y selección de rango.[14]

Switch y Toggle

- **Propósito y Casos de Uso:** Ambos son controles de dos estados. El Switch es un control deslizante de encendido/apagado, comúnmente utilizado para activar o desactivar configuraciones. El Toggle es un botón que puede estar presionado o no, y el ToggleGroup permite agrupar varios de estos botones para selecciones de formato

(como negrita, cursiva) o filtros.[14]

Textarea

- **Propósito y Casos de Uso:** Un campo de entrada para texto de múltiples líneas, basado en el elemento nativo <textarea>. Admite estados de disabled y hasError.[14]

Sección 6: Componentes de UI Centrales

Estos componentes son los bloques de construcción fundamentales para estructurar el layout y la interfaz de cualquier aplicación o dashboard.

La Fundación de Radix UI

Es importante destacar que muchos de los componentes de UI más complejos de Tremor no se construyen desde cero. En su lugar, aprovechan las primitivas de componentes sin estilo, accesibles y robustas de Radix UI.[15] Tremor añade la capa de estilo con Tailwind CSS y los compone en elementos listos para dashboards. Para los desarrolladores, esto significa que para casos de uso avanzados que requieran un control de bajo nivel sobre la accesibilidad (aria-*), el manejo de eventos o la gestión del estado que no están expuestos directamente por las props de Tremor, pueden consultar la documentación de Radix UI para el componente subyacente. Esto proporciona una vía de escape para una personalización y comprensión más profundas.

Accordion (Acordeón)

- **Propósito y Casos de Uso:** Un conjunto de paneles apilados verticalmente que revelan u ocultan su contenido cuando se hace clic en sus cabeceras. Es útil para gestionar grandes cantidades de información en un espacio limitado, como en secciones de preguntas frecuentes (FAQ).[15]

Badge (Insignia)

- **Propósito y Casos de Uso:** Un pequeño elemento de UI utilizado para resaltar estados, etiquetas o conteos. Se usa comúnmente en tablas o listas para indicar el estado de un elemento (p. ej., "Activo", "Error").[18]

Button (Botón)

- **Propósito y Casos de Uso:** Un control interactivo para que los usuarios inicien una acción. Tremor proporciona variantes de estilo (primario, secundario, etc.) y tamaños.[5]

Callout (Llamada de Atención)

- **Propósito y Casos de Uso:** Un bloque de contenido destacado diseñado para atraer la atención del usuario hacia información importante, como advertencias, consejos o notificaciones de éxito.[5]

Card (Tarjeta)

- **Propósito y Casos de Uso:** Un contenedor fundamental para agrupar contenido relacionado. Las tarjetas son los bloques de construcción básicos para dashboards, utilizados para mostrar KPIs, formularios o secciones de contenido de forma modular.[19]

Dialog (Diálogo)

- **Propósito y Casos de Uso:** Una ventana modal que se superpone al contenido principal de la página, requiriendo la interacción del usuario. Se utiliza para acciones críticas, confirmaciones o para mostrar información detallada sin abandonar el contexto actual.[16]

Divider (Divisor)

- **Propósito y Casos de Uso:** Una línea horizontal o vertical utilizada para separar y organizar visualmente el contenido dentro de un layout.[16]

Drawer (Cajón Lateral)

- **Propósito y Casos de Uso:** Un panel que se desliza desde el borde de la pantalla (generalmente izquierdo o derecho). Es útil para menús de navegación, filtros o formularios que no necesitan estar visibles permanentemente.[16]

Popover

- **Propósito y Casos de Uso:** Un pequeño contenedor de contenido que aparece junto a un elemento disparador cuando se hace clic en él. A diferencia de un tooltip, puede contener contenido interactivo como botones o formularios.[16]

Table (Tabla)

- **Propósito y Casos de Uso:** Se utiliza para mostrar conjuntos de datos estructurados en filas y columnas de manera eficiente. Los componentes de tabla de Tremor están diseñados para ser responsivos, a menudo utilizando un contenedor que permite el desplazamiento horizontal en pantallas pequeñas.[18]

Tabs y TabNavigation

- **Propósito y Casos de Uso:** Ambos componentes organizan el contenido en vistas separadas. Tabs se utiliza para alternar entre diferentes paneles de contenido dentro de la misma página. TabNavigation está diseñado específicamente para la navegación,

donde cada pestaña es un enlace que lleva a una ruta o página diferente.[18]

Toast

- **Propósito y Casos de Uso:** Una notificación ligera y no intrusiva que aparece brevemente para proporcionar retroalimentación sobre una acción (p. ej., "Archivo guardado correctamente"). Vienen en diferentes variantes (info, success, warning, error, loading) para comunicar el estado adecuado.[17]

Tooltip

- **Propósito y Casos de Uso:** Una pequeña ventana emergente que muestra información descriptiva cuando el usuario pasa el cursor sobre un elemento o lo enfoca. Es ideal para aclarar la función de un ícono o proporcionar detalles adicionales sin abarrotar la interfaz.[17]

Sección 7: Funciones de Utilidad y Ayudantes

Además de los componentes de UI, Tremor proporciona un conjunto de funciones de utilidad para simplificar tareas comunes de desarrollo.[5]

- **chartUtils:** Probablemente un conjunto de funciones auxiliares relacionadas con la manipulación o el formato de datos para los componentes de gráficos.
- **cx:** Una utilidad común en el ecosistema de React y Tailwind para construir cadenas de nombres de clase condicionalmente. Permite combinar clases de forma limpia y legible. Por ejemplo: cx('clase-base', isActive && 'clase-activa', hasError? 'clase-error' : 'clase-normal').
- **focusInput, hasErrorInput, focusRing:** Estas utilidades probablemente encapsulan la lógica de estilo para los estados de los inputs, como aplicar un anillo de enfoque (focusRing) o estilos de error (hasErrorInput), promoviendo la consistencia en los formularios.

Parte III: Ecosistema y Mejores Prácticas

Esta parte final sintetiza la información para proporcionar una guía de alto nivel sobre la construcción de aplicaciones con Tremor, abordando las mejores prácticas para la creación de layouts de dashboards.

Sección 8: El Ecosistema Ampliado de Tremor

Para maximizar la productividad, los desarrolladores deben aprovechar todo el ecosistema que Tremor ofrece más allá de su biblioteca de componentes principal.

- **Tremor Blocks:** En lugar de construir interfaces complejas desde cero, los desarrolladores pueden utilizar "Blocks". Estos son componentes compuestos y listos para producción que se pueden copiar y pegar directamente en una aplicación. Ejemplos incluyen tarjetas de KPI completas con gráficos de chispa, tablas de datos con insignias de estado y filtros avanzados.[5] El uso de bloques acelera drásticamente el desarrollo al proporcionar soluciones a patrones de UI comunes.
- **Templates:** Para proyectos más grandes, Tremor ofrece plantillas de aplicaciones completas. Estas plantillas, como el "Dashboard Template" o el "SaaS Template", proporcionan una estructura de proyecto completa, con enrutamiento, layouts y componentes ya integrados.[2] Son puntos de partida invaluables que ahorran semanas de trabajo de configuración inicial y demuestran cómo estructurar una aplicación a gran escala con Tremor.
- **Figma UI Kit:** La colaboración efectiva entre diseño y desarrollo es crucial. El Figma UI Kit de Tremor cierra la brecha al proporcionar a los diseñadores un conjunto de componentes que se corresponden directamente con la biblioteca de React.[5] Esto asegura que las maquetas de diseño sean factibles de implementar, reduce la ambigüedad durante la transferencia y garantiza la consistencia visual entre el diseño y el producto final.
- **Iconos (Remix Icon):** Tremor estandariza el uso de la biblioteca Remix Icon. Es un conjunto de iconos de código abierto, completo y bien diseñado, licenciado bajo la permisiva Apache 2.0.[5] Al saber que esta es la biblioteca de iconos predeterminada, los desarrolladores pueden buscar y utilizar fácilmente cualquier icono del conjunto, manteniendo un estilo visual coherente en toda la aplicación.

Sección 9: Principios de Layout y Diseño Responsivo

Aunque la guía oficial "Your First Dashboard" no estaba accesible durante la investigación [20], es posible inferir las mejores prácticas de layout y diseño responsivo analizando la implementación de los propios componentes de Tremor.

Prácticas Inferidas a partir de la Implementación de Componentes

La ausencia de una guía de layout explícita sugiere que Tremor no impone un sistema de layout rígido. En cambio, capacita a los desarrolladores para que utilicen el poder y la flexibilidad de las utilidades de layout de Tailwind CSS (principalmente Flexbox y Grid) para construir sus propias estructuras responsivas.

- **Manejo Responsivo de Tablas:** El componente Table de Tremor ofrece una visión clara de un enfoque pragmático para los datos responsivos. El componente TableRoot envuelve la tabla en un div con las clases w-full overflow-auto whitespace-nowrap.[19] En lugar de intentar reajustar las columnas de la tabla de una manera compleja que podría comprometer la legibilidad, esta técnica hace que la tabla sea desplazable horizontalmente en pantallas pequeñas. Este es un patrón de diseño común y efectivo que preserva la integridad de los datos tabulares en dispositivos móviles, asegurando que toda la información permanezca accesible.
- **Composición con Utilidades de Layout de Tailwind:** Los ejemplos de composición de componentes demuestran que la estructura del layout se deja en manos del desarrollador, utilizando las herramientas estándar de Tailwind. Por ejemplo, un ejemplo del componente Card utiliza un <ul> como contenedor con clases de Flexbox (flex list-none flex-col gap-4) para organizar una lista de tarjetas verticalmente con un espaciado consistente.[19] Esto indica que la filosofía de Tremor es proporcionar los bloques de construcción (los componentes), mientras que el "cemento" que los une (el layout) proviene directamente de las utilidades de Flexbox y Grid de Tailwind.

Recomendaciones para la Construcción de Layouts

Basado en este análisis, la principal recomendación para construir layouts complejos y responsivos con Tremor es tener un dominio sólido de las capacidades de layout de Tailwind CSS.

1. **Utilice Flexbox y Grid:** Aproveche las utilidades de Flexbox (flex, justify-*, items-*) y CSS Grid (grid, grid-cols-*, gap-*) de Tailwind para crear la estructura principal de sus

dashboards. Grid es particularmente útil para el layout general de la página (por ejemplo, una barra lateral y un área de contenido principal), mientras que Flexbox es excelente para alinear elementos dentro de los componentes.

2. **Adopte un Enfoque "Mobile-First":** Diseñe sus layouts primero para pantallas pequeñas y luego use los modificadores responsivos de Tailwind (como sm:, md:, lg:) para adaptar el layout a pantallas más grandes. Por ejemplo, una serie de tarjetas de KPI podría apilarse verticalmente en móviles (flex-col) y luego cambiar a una cuadrícula de varias columnas en pantallas de escritorio (md:grid md:grid-cols-3).
3. **Aproveche los Componentes de Layout Primitivos:** Aunque Tremor no tiene componentes de "Layout" dedicados, componentes como Card y Divider son fundamentales para crear separación visual y estructura. Utilícelos como contenedores modulares dentro de sus sistemas de Flexbox y Grid.
4. **Siga los Patrones de Componentes:** Observe cómo se construyen los componentes de Tremor y los "Blocks" para obtener pistas sobre patrones de layout efectivos. La combinación de componentes y utilidades de Tailwind en estos ejemplos preconstruidos representa un conjunto de mejores prácticas respaldadas por los creadores de la biblioteca.

En esencia, Tremor proporciona los componentes de UI de alta calidad, pero la responsabilidad de orquestarlos en un dashboard cohesivo y responsivo recae en la habilidad del desarrollador para aplicar eficazmente las potentes y expresivas utilidades de layout de Tailwind CSS.

Obras citadas

1. tremorlabs/tremor: Copy & Paste React components to build modern web applications. - GitHub, fecha de acceso: octubre 9, 2025, https://github.com/tremorlabs/tremor
2. Tremor.so, fecha de acceso: octubre 9, 2025, https://tremor.so/
3. tremorlabs/tremor-npm: React components to build charts and dashboards - GitHub, fecha de acceso: octubre 9, 2025, https://github.com/tremorlabs/tremor-npm
4. tremorlabs/template-dashboard-oss: Free open-source dashboard template by Tremor. - GitHub, fecha de acceso: octubre 9, 2025, https://github.com/tremorlabs/template-dashboard-oss
5. Installation - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/getting-started/installation
6. Installation • Docs - Tremor NPM, fecha de acceso: octubre 9, 2025, https://npm.tremor.so/docs/getting-started/installation
7. How the integrate Tremor to build interactive dashboards | supastarter - SaaS starter kit for Next.js, Nuxt and SvelteKit, fecha de acceso: octubre 9, 2025, https://supastarter.dev/blog/tremor-integration
8. tremor - GitHub, fecha de acceso: octubre 9, 2025, https://github.com/tremorlabs
9. Build dashboards with React/Next.js and Tailwind CSS! New Tremor v2.0 release -

Reddit, fecha de acceso: octubre 9, 2025, https://www.reddit.com/r/programming/comments/11v1ap5/build_dashboards_with_reactnextjs_and_tailwind/
10. Build Dashboards Fast with Tremor - RedwoodJS Docs, fecha de acceso: octubre 9, 2025, https://docs.redwoodjs.com/docs/5.x/how-to/build-dashboards-fast-with-tremor/
11. Tailwind breakpoints are working unexpectedly · tailwindlabs tailwindcss · Discussion #13600 - GitHub, fecha de acceso: octubre 9, 2025, https://github.com/tailwindlabs/tailwindcss/discussions/13600
12. How to Build Data Dashboards in React with Tremor & Tailwind - Space Jelly, fecha de acceso: octubre 9, 2025, https://spacejelly.dev/posts/how-to-build-data-dashboards-in-react-with-tremor-tailwind
13. Area Chart - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/visualizations/area-chart
14. Calendar - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/inputs/calendar
15. Accordion - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/ui/accordion
16. Dialog - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/ui/dialog
17. Toast - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/ui/toast
18. Table - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/ui/table
19. Card - Tremor, fecha de acceso: octubre 9, 2025, https://tremor.so/docs/ui/card
20. fecha de acceso: diciembre 31, 1969, https://tremor.so/docs/getting-started/your-first-dashboard