File: client/src/components/ui/chart.tsx

Type: tsx | Size: 10481 bytes

```tsx
"use client"

import * as React from "react"
import * as RechartsPrimitive from "recharts"

import { cn } from "@/lib/utils"

// Format: { THEME_NAME: CSS_SELECTOR }
const THEMES = { light: "", dark: ".dark" } as const

export type ChartConfig = {
  [k in string]: {
    label?: React.ReactNode
    icon?: React.ComponentType
  } & (
    | { color?: string; theme?: never }
    | { color?: never; theme: Record<keyof typeof THEMES, string> }
  )
}

type ChartContextProps = {
  config: ChartConfig
}

const ChartContext = React.createContext<ChartContextProps | null>(null)

function useChart() {
  const context = React.useContext(ChartContext)

  if (!context) {
    throw new Error("useChart must be used within a <ChartContainer />")
  }

  return context
}

const ChartContainer = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> & {
    config: ChartConfig
    children: React.ComponentProps<
      typeof RechartsPrimitive.ResponsiveContainer
    >["children"]
  }
>(({ id, className, children, config, ...props }, ref) => {
  const uniqueId = React.useId()
  const chartId = `chart-${id || uniqueId.replace(/:/g, "")}`

  return (
    <ChartContext.Provider value={{ config }}>
      <div
        data-chart={chartId}
        ref={ref}
        className={cn(
          "flex aspect-video justify-center text-xs [&_.recharts-cartesian-axis-tick_text]:fill-muted-foregrou...
          className
        )}
        {...props}
      >
        <ChartStyle id={chartId} config={config} />
        <RechartsPrimitive.ResponsiveContainer>
          {children}
        </RechartsPrimitive.ResponsiveContainer>
      </div>
    </ChartContext.Provider>
  )
})
ChartContainer.displayName = "Chart"

const ChartStyle = ({ id, config }: { id: string; config: ChartConfig }) => {
```

```
  const colorConfig = Object.entries(config).filter(
    ([, config]) => config.theme || config.color
  )

  if (!colorConfig.length) {
    return null
  }

  return (
    <style
      dangerouslySetInnerHTML={{
        __html: Object.entries(THEMES)
          .map(
            ([theme, prefix]) => `
${prefix} [data-chart=${id}] {
${colorConfig
  .map(([key, itemConfig]) => {
    const color =
      itemConfig.theme?.[theme as keyof typeof itemConfig.theme] ||
      itemConfig.color
    return color ? `  --color-${key}: ${color};` : null
  })
  .join("\n")}
}
`
          )
          .join("\n"),
      }}
    />
  )
}

const ChartTooltip = RechartsPrimitive.Tooltip

const ChartTooltipContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<typeof RechartsPrimitive.Tooltip> &
    React.ComponentProps<"div"> & {
      hideLabel?: boolean
      hideIndicator?: boolean
      indicator?: "line" | "dot" | "dashed"
      nameKey?: string
      labelKey?: string
    }
>(
  (
    {
      active,
      payload,
      className,
      indicator = "dot",
      hideLabel = false,
      hideIndicator = false,
      label,
      labelFormatter,
      labelClassName,
      formatter,
      color,
      nameKey,
      labelKey,
    },
    ref
  ) => {
    const { config } = useChart()

    const tooltipLabel = React.useMemo(() => {
      if (hideLabel || !payload?.length) {
        return null
      }

      const [item] = payload
      const key = `${labelKey || item?.dataKey || item?.name || "value"}`
      const itemConfig = getPayloadConfigFromPayload(config, item, key)
```

```
  const value =
    !labelKey && typeof label === "string"
      ? config[label as keyof typeof config]?.label || label
      : itemConfig?.label

  if (labelFormatter) {
    return (
      <div className={cn("font-medium", labelClassName)}>
        {labelFormatter(value, payload)}
      </div>
    )
  }

  if (!value) {
    return null
  }

  return <div className={cn("font-medium", labelClassName)}>{value}</div>
}, [
  label,
  labelFormatter,
  payload,
  hideLabel,
  labelClassName,
  config,
  labelKey,
])

if (!active || !payload?.length) {
  return null
}

const nestLabel = payload.length === 1 && indicator !== "dot"

return (
  <div
    ref={ref}
    className={cn(
      "grid min-w-[8rem] items-start gap-1.5 rounded-lg border border-border/50 bg-background px-2.5 py-1....
      className
    )}
  >
    {!nestLabel ? tooltipLabel : null}
    <div className="grid gap-1.5">
      {payload.map((item, index) => {
        const key = `${nameKey || item.name || item.dataKey || "value"}`
        const itemConfig = getPayloadConfigFromPayload(config, item, key)
        const indicatorColor = color || item.payload.fill || item.color

        return (
          <div
            key={item.dataKey}
            className={cn(
              "flex w-full flex-wrap items-stretch gap-2 [&>svg]:h-2.5 [&>svg]:w-2.5 [&>svg]:text-muted-fo...
              indicator === "dot" && "items-center"
            )}
          >
            {formatter && item?.value !== undefined && item.name ? (
              formatter(item.value, item.name, item, index, item.payload)
            ) : (
              <>
                {itemConfig?.icon ? (
                  <itemConfig.icon />
                ) : (
                  !hideIndicator && (
                    <div
                      className={cn(
                        "shrink-0 rounded-[2px] border-[--color-border] bg-[--color-bg]",
                        {
                          "h-2.5 w-2.5": indicator === "dot",
                          "w-1": indicator === "line",
                          "w-0 border-[1.5px] border-dashed bg-transparent":
                            indicator === "dashed",
```

```
                            "my-0.5": nestLabel && indicator === "dashed",
                          }
                        )}
                        style={
                          {
                            "--color-bg": indicatorColor,
                            "--color-border": indicatorColor,
                          } as React.CSSProperties
                        }
                      />
                    )
                  )}
                  <div
                    className={cn(
                      "flex flex-1 justify-between leading-none",
                      nestLabel ? "items-end" : "items-center"
                    )}
                  >
                    <div className="grid gap-1.5">
                      {nestLabel ? tooltipLabel : null}
                      <span className="text-muted-foreground">
                        {itemConfig?.label || item.name}
                      </span>
                    </div>
                    {item.value && (
                      <span className="font-mono font-medium tabular-nums text-foreground">
                        {item.value.toLocaleString()}
                      </span>
                    )}
                  </div>
                </>
              )}
            </div>
          )
        })}
      </div>
    </div>
  )
}
)
ChartTooltipContent.displayName = "ChartTooltip"

const ChartLegend = RechartsPrimitive.Legend

const ChartLegendContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> &
    Pick<RechartsPrimitive.LegendProps, "payload" | "verticalAlign"> & {
      hideIcon?: boolean
      nameKey?: string
    }
>(
  (
    { className, hideIcon = false, payload, verticalAlign = "bottom", nameKey },
    ref
  ) => {
    const { config } = useChart()

    if (!payload?.length) {
      return null
    }

    return (
      <div
        ref={ref}
        className={cn(
          "flex items-center justify-center gap-4",
          verticalAlign === "top" ? "pb-3" : "pt-3",
          className
        )}
      >
        {payload.map((item) => {
          const key = `${nameKey || item.dataKey || "value"}`
```

```
          const itemConfig = getPayloadConfigFromPayload(config, item, key)

          return (
            <div
              key={item.value}
              className={cn(
                "flex items-center gap-1.5 [&>svg]:h-3 [&>svg]:w-3 [&>svg]:text-muted-foreground"
              )}
            >
              {itemConfig?.icon && !hideIcon ? (
                <itemConfig.icon />
              ) : (
                <div
                  className="h-2 w-2 shrink-0 rounded-[2px]"
                  style={{
                    backgroundColor: item.color,
                  }}
                />
              )}
              {itemConfig?.label}
            </div>
          )
        })}
      </div>
    )
  }
)
ChartLegendContent.displayName = "ChartLegend"

// Helper to extract item config from a payload.
function getPayloadConfigFromPayload(
  config: ChartConfig,
  payload: unknown,
  key: string
) {
  if (typeof payload !== "object" || payload === null) {
    return undefined
  }

  const payloadPayload =
    "payload" in payload &&
    typeof payload.payload === "object" &&
    payload.payload !== null
      ? payload.payload
      : undefined

  let configLabelKey: string = key

  if (
    key in payload &&
    typeof payload[key as keyof typeof payload] === "string"
  ) {
    configLabelKey = payload[key as keyof typeof payload] as string
  } else if (
    payloadPayload &&
    key in payloadPayload &&
    typeof payloadPayload[key as keyof typeof payloadPayload] === "string"
  ) {
    configLabelKey = payloadPayload[
      key as keyof typeof payloadPayload
    ] as string
  }

  return configLabelKey in config
    ? config[configLabelKey]
    : config[key as keyof typeof config]
}

export {
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
  ChartLegend,
```

```
  ChartLegendContent,
  ChartStyle,
}
```