Recharts Visualization System Documentation

Overview

This document provides comprehensive guidance on our Apple-inspired chart system built on top of Recharts. The system is designed with a focus on:

- **Consistent styling** across all chart types
- **Theme integration** for light and dark mode support
- **Gradient-enhanced visuals** for a modern, 3D look
- **Flexible color palettes** for different data visualization needs
- **Reusable components** for rapid implementation

Table of Contents

Architecture

The chart system is structured in three layers:

1. **Style System**: A hook-based approach (`useChartStyles`) that manages colors, gradients, and styling preferences
2. **Base Components**: Reusable chart components for common chart types (Bar, Line, Pie, Radar)
3. **Business Components**: App-specific implementations that use base components

```
src/
├── dashboard/
│   └── components/
│       └── dashboard/
│           └── charts/
│               └── [Business chart components]
├── components/
│   └── ui/
│       └── charts/
│           ├── BaseBarChart.tsx
│           ├── BaseLineChart.tsx
│           ├── BasePieChart.tsx
│           └── BaseRadarChart.tsx
```

```
└── hooks/
    └── useChartStyles.ts
```

useChartStyles Hook

The `useChartStyles` hook is the foundation of the chart system, providing access to colors, gradients, and style options.

Import

```tsx
import { useChartStyles } from "@/hooks/useChartStyles";
```

Usage

```tsx
const {
  chartColors,
  getColorPalette,
  getGradientDefs,
  getRadarGradients,
  chartDefaults
} = useChartStyles({
  variant: 'categorical',
  opacity: 0.8,
  useGradient: true
});
```

Options

Parameter	Type	Default	Description
`variant`	'primary' | 'sequential' | 'categorical' | 'divergent'	'primary'	Color palette type
`opacity`	number	0.8	Base opacity for gradients
`useGradient`	boolean	true	Whether to use gradients
`colorCount`	number	8	Number of colors to generate

Return Values

- **chartColors**: Object containing all theme-defined chart colors
- **getColorPalette(count?)**: Function returning an array of colors based on variant

- **getGradientDefs(colors?)**: Function returning gradient definitions for SVG
- **getRadarGradients(colors?)**: Function for radar chart gradients
- **chartDefaults**: Object containing Apple-inspired styling defaults
- **isDark**: Boolean indicating if dark mode is active

Color Variants

- **Primary**: All 8 colors in the palette
- **Sequential**: Colors from cool to warm (useful for progression)
- **Categorical**: Evenly distributed distinct colors (best for categories)
- **Divergent**: Blue to red spectrum (useful for comparisons)

Base Chart Components

The system provides five base chart components that handle common chart types.

BaseBarChart

A component for rendering bar charts, supporting both vertical and horizontal layouts.

```
import { BaseBarChart } from "@/components/ui/charts/core/BaseBarChart";

<BaseBarChart
  data={data}
  xKey="category"
  yKeys={["value1", "value2"]}
  stacked={false}
  layout="horizontal"
  height={300}
  variant="categorical"
/>
```

Props

Prop	Type	Default	Description
`data`	any[]	(required)	Array of data objects
`xKey`	string	(required)	Key for X-axis values
`yKeys`	string[]	(required)	Keys for Y-axis values
`stacked`	boolean	false	Whether bars should be stacked
`layout`	'vertical' | 'horizontal'	'horizontal'	Chart orientation

Prop	Type	Default	Description
barSize	number	40	Width of bars in pixels
showLegend	boolean	true	Whether to show the legend
height	number | string	300	Chart height
margin	object	{ top: 20, right: 30, left: 20, bottom: 30 }	Chart margins
customLegend	ReactNode	undefined	Custom legend component
variant	'primary' | 'sequential' | 'categorical' | 'divergent'	'categorical'	Color variant
domain	[number, number]	undefined	Y-axis domain

BaseLineChart

A component for rendering line charts with optional curve smoothing and dots.

```
import { BaseLineChart } from
"@/components/ui/charts/core/BaseLineChart";

<BaseLineChart
  data={data}
  xKey="date"
  yKeys={["metric1", "metric2"]}
  curved={true}
  showDots={true}
  height={300}
/>
```

Props

Prop	Type	Default	Description
data	any[]	(required)	Array of data objects
xKey	string	(required)	Key for X-axis values
yKeys	string[]	(required)	Keys for Y-axis values

Prop	Type	Default	Description
height	number | string	300	Chart height
margin	object	{ top: 20, right: 30, left: 20, bottom: 30 }	Chart margins
showLegend	boolean	true	Whether to show the legend
customLegend	ReactNode	undefined	Custom legend component
variant	'primary' | 'sequential' | 'categorical' | 'divergent'	'categorical'	Color variant
domain	[number, number]	undefined	Y-axis domain
curved	boolean	true	Whether to smooth the lines
showDots	boolean	true	Whether to show data points
customDot	ReactNode	undefined	Custom dot component

BasePieChart

A component for rendering pie and donut charts with customizable labels.

```
import { BasePieChart } from "@/components/ui/charts/core/BasePieChart";

<BasePieChart
  data={data}
  dataKey="value"
  nameKey="name"
  innerRadius={50}
  outerRadius={90}
  showLabels={true}
  labelType="name-percent"
/>
```

Props

Prop	Type	Default	Description
data	Array<{ name: string; value: number; [key: string]: any }>	(required)	Array of data objects

Prop	Type	Default	Description
dataKey	string	'value'	Key for segment values
nameKey	string	'name'	Key for segment names
height	number | string	300	Chart height
innerRadius	number	0	Inner radius (0 for pie, >0 for donut)
outerRadius	number	90	Outer radius
paddingAngle	number	0	Angle between segments
showLegend	boolean	true	Whether to show the legend
customLegend	ReactNode	undefined	Custom legend component
variant	'primary' | 'sequential' | 'categorical' | 'divergent'	'categorical'	Color variant
showLabels	boolean	true	Whether to show labels
labelType	'name' | 'value' | 'percent' | 'name-percent'	'name-percent'	Type of label to display

BaseRadarChart

A component for rendering radar/spider charts.

```
import { BaseRadarChart } from
"@/components/ui/charts/core/BaseRadarChart";

<BaseRadarChart
  data={data}
  dataKeys={["metric1", "metric2", "metric3"]}
  labelKey="axis"
  outerRadius={90}
  domain={[0, 100]}
/>
```

Props

Prop	Type	Default	Description
data	any[]	(required)	Array of data objects
dataKeys	string[]	(required)	Keys for radar metrics
labelKey	string	(required)	Key for axis labels
height	number | string	300	Chart height
showLegend	boolean	true	Whether to show the legend
customLegend	ReactNode	undefined	Custom legend component
variant	'primary' | 'sequential' | 'categorical' | 'divergent'	'categorical'	Color variant
outerRadius	number	90	Radar chart radius
domain	[number	'auto'	'dataMin'

BaseAreaChart

A component for rendering area charts with optional stacking and curve smoothing.

```
import { BaseAreaChart } from
"@/components/ui/charts/core/BaseAreaChart";

<BaseAreaChart
  data={data}
  xKey="date"
  yKeys={["revenue", "costs"]}
  stacked={true}
  curved={true}
  showDots={false}
  height={300}
/>
```

Props

Prop	Type	Default	Description
data	any[]	(required)	Array of data objects
xKey	string	(required)	Key for X-axis values

Prop	Type	Default	Description
yKeys	string[]	(required)	Keys for Y-axis values
height	number | string	300	Chart height
margin	object	{ top: 20, right: 30, left: 20, bottom: 30 }	Chart margins
showLegend	boolean	true	Whether to show the legend
customLegend	ReactNode	undefined	Custom legend component
variant	'primary' | 'sequential' | 'categorical' | 'divergent'	'categorical'	Color variant
domain	[number	'auto'	'dataMin'
stacked	boolean	false	Whether areas should be stacked
curved	boolean	true	Whether to smooth the lines
showDots	boolean	false	Whether to show data points
customDot	ReactNode	undefined	Custom dot component

Implementing Charts

Basic Implementation

1. Import the appropriate base chart component
2. Pass your data and configuration props
3. All components include ResponsiveContainer, so they automatically adjust to their parent container

```
import { Card, CardHeader, CardTitle, CardContent } from
"@/components/ui/core/Card";
import { BaseBarChart } from "@/components/ui/charts/core/BaseBarChart";

export function SalesReport() {
  const data = [
    { month: 'Jan', sales: 100, profit: 30 },
    { month: 'Feb', sales: 200, profit: 50 },
    { month: 'Mar', sales: 150, profit: 40 },
    // ...more data
```

```
  ];

  return (
    <Card>
      <CardHeader>
        <CardTitle>Monthly Sales & Profit</CardTitle>
      </CardHeader>
      <CardContent>
        <BaseBarChart
          data={data}
          xKey="month"
          yKeys={["sales", "profit"]}
          height={300}
          variant="primary"
        />
      </CardContent>
    </Card>
  );
}
```

Advanced Implementation

For more complex charts or custom business logic:

1. Create a component that encapsulates your specific requirements
2. Use the base chart component inside your component
3. Handle any data transformations or special interactions

```
import { useState, useEffect } from "react";
import { Card, CardHeader, CardTitle, CardContent } from
"@/components/ui/core/Card";
import { BaseLineChart } from
"@/components/ui/charts/core/BaseLineChart";
import { fetchSalesData } from "@/lib/api";

export function SalesTrend() {
  const [data, setData] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const loadData = async () => {
      try {
        const response = await fetchSalesData();
        setData(response);
      } catch (err) {
        setError(err);
      } finally {
        setIsLoading(false);
      }
```

```
    };

    loadData();
  }, []);

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error loading data</div>;

  return (
    <Card>
      <CardHeader>
        <CardTitle>Sales Trend Analysis</CardTitle>
      </CardHeader>
      <CardContent>
        <BaseLineChart
          data={data}
          xKey="date"
          yKeys={["revenue", "forecast"]}
          height={300}
          variant="sequential"
          domain={[0, 'auto']}
          margin={{ top: 20, right: 30, left: 20, bottom: 30 }}
        />
      </CardContent>
    </Card>
  );
}
```

Customization Guide

Custom Colors

You can override the default color palette in your theme.css file:

```
:root {
  --chart-deep-ocean: hsl(200 95% 19%);
  --chart-midnight-sky: hsl(217 44% 33%);
  --chart-stormy-violet: hsl(261 16% 36%);
  --chart-wild-berry: hsl(312 33% 47%);
  --chart-rosewood: hsl(327 54% 56%);
  --chart-sunset-blush: hsl(350 86% 67%);
  --chart-autumn-leaf: hsl(19 100% 64%);
  --chart-golden-sun: hsl(39 100% 53%);
}
```

Custom Gradients

To customize gradient appearance:

```
const { getGradientDefs } = useChartStyles({
  opacity: 0.9, // Increase opacity
});
```

Or create your own gradient:

```
<defs>
  <linearGradient id="customGradient" x1="0" y1="0" x2="0" y2="1">
    <stop offset="5%" stopColor="#8884d8" stopOpacity={0.8}/>
    <stop offset="95%" stopColor="#8884d8" stopOpacity={0.1}/>
  </linearGradient>
</defs>
```

Custom Legend

You can provide a custom legend component:

```
const CustomLegend = (props) => {
  const { payload } = props;

  return (
    <div className="flex justify-center gap-4 mt-4 text-sm">
      {payload.map((entry, index) => (
        <div key={`item-${index}`} className="flex items-center">
          <span
            className="inline-block w-3 h-3 mr-2 rounded-full"
            style={{ backgroundColor: entry.color }}
          />
          <span>{entry.value}</span>
        </div>
      ))}
    </div>
  );
};

<BaseLineChart
  // ...other props
  customLegend={CustomLegend}
/>
```

Custom Tooltips

To customize tooltips, override the tooltipStyle in useChartStyles:

```
// Extend the useChartStyles hook
const useCustomChartStyles = (options) => {
  const styles = useChartStyles(options);

  return {
    ...styles,
    chartDefaults: {
      ...styles.chartDefaults,
      tooltipStyle: {
        contentStyle: {
          ...styles.chartDefaults.tooltipStyle.contentStyle,
          backgroundColor: "rgba(0, 0, 0, 0.8)",
          color: "#ffffff",
          borderRadius: "4px",
        },
        itemStyle: {
          ...styles.chartDefaults.tooltipStyle.itemStyle,
          color: "#ffffff",
        },
        labelStyle: {
          ...styles.chartDefaults.tooltipStyle.labelStyle,
          color: "#ffffff",
          fontWeight: "bold",
        }
      }
    }
  };
};
```

Theming

The chart system automatically adapts to light and dark themes, reading from CSS variables.

Theme Integration

The useChartStyles hook accesses these CSS variables:

```
--chart-deep-ocean
--chart-midnight-sky
--chart-stormy-violet
--chart-wild-berry
--chart-rosewood
--chart-sunset-blush
--chart-autumn-leaf
--chart-golden-sun
--chart-grid
--chart-axis
--chart-label
--chart-tick
```

Dark Mode Support

In dark mode, these variables are automatically adjusted:

```
.dark {
  --chart-grid: hsl(220 10% 25%);
  --chart-axis: hsl(220 10% 30%);
  --chart-label: hsl(220 10% 90%);
  --chart-tick: hsl(220 10% 70%);
}
```

The hook detects dark mode and adjusts tooltip styles accordingly.

Best Practices

Data Formatting

- Ensure data is properly formatted before passing to chart components
- For time series data, preprocess dates into the right format
- Always verify data integrity with console logs during development

Performance Considerations

- Limit the number of data points for optimal rendering (< 100 points for complex charts)
- Use memoization for expensive data transformations
- Consider lazy loading charts for pages with multiple visualizations

Accessibility

- Use color combinations with adequate contrast
- Add meaningful aria labels to chart containers
- Ensure interactive elements are keyboard accessible

Responsive Design

- Always use ResponsiveContainer for chart components
- Set appropriate min/max heights to prevent layout shifts
- Test charts across different screen sizes

Hiding the Legend

All chart components support hiding the legend by setting the `showLegend` prop to `false`:

```
<BaseBarChart
  data={salesData}
  xKey="category"
```

```
  yKeys={["revenue"]}
  showLegend={false} // Hide the legend
/>
```

This is useful when:

- The chart has only one data series and the legend is redundant
- You want to save vertical space in the UI
- The labels on the chart already provide enough context
- You're showing multiple small charts in a dashboard

ResponsiveContainer

All chart components are wrapped in Recharts' `ResponsiveContainer` which:

- Automatically resizes the chart to fit its parent container
- Maintains aspect ratio during resizing
- Responds to window size changes

```
// Inside each base chart component
<div style={{ width: '100%', height }}>
  <ResponsiveContainer width="100%" height="100%">
    {/* Chart content */}
  </ResponsiveContainer>
</div>
```

To ensure proper sizing:

- Always set a specific `height` prop (default is 300px)
- Ensure the parent container has a defined width
- For small multiples or grid layouts, use CSS grid or flexbox for layout

Interactive Legend

All charts include an interactive legend with hover effects:

- Hovering over a legend item highlights the corresponding series
- Non-highlighted series are dimmed with reduced opacity
- This helps users focus on specific data series in complex charts

Code Organization

- Keep data transformation logic separate from chart rendering
- Create specific chart components for reused visualizations
- Document chart props and data requirements in comments

Examples

Example 1: Team Performance Trend

```
import { Card, CardHeader, CardTitle, CardDescription, CardContent }
from "@/components/ui/core/Card";
import { BaseLineChart } from
"@/components/ui/charts/core/BaseLineChart";

export function TeamPerformanceTrend() {
  const data = [
    {
      period: "Q1 2023",
      Managers: 3.8,
      "Product Design": 4.2,
      "Design Technology": 4.2,
      Research: 4.2,
    },
    // more data...
  ];

  // Get team names
  const teamNames = Object.keys(data[0]).filter(key => key !==
'period');

  return (
    <Card className="bg-white/80 backdrop-blur-sm border border-slate-
200 shadow-sm">
      <CardHeader>
        <CardTitle>Performance Trends</CardTitle>
        <CardDescription>
          Team performance evolution over time
        </CardDescription>
      </CardHeader>
      <CardContent>
        <BaseLineChart
          data={data}
          xKey="period"
          yKeys={teamNames}
          height={320}
          domain={[0, 5]}
          curved={true}
        />
      </CardContent>
    </Card>
  );
}
```

Example 2: Skill Distribution

```
import { Card, CardHeader, CardTitle, CardDescription, CardContent }
from "@/components/ui/core/Card";
import { BaseRadarChart } from
"@/components/ui/charts/core/BaseRadarChart";

export function SkillDistribution() {
  const data = [
    { skill: 'Communication', Team1: 90, Team2: 60, Team3: 75 },
    { skill: 'Technical', Team1: 70, Team2: 85, Team3: 65 },
    { skill: 'Leadership', Team1: 60, Team2: 70, Team3: 85 },
    { skill: 'Collaboration', Team1: 80, Team2: 75, Team3: 70 },
    { skill: 'Problem Solving', Team1: 75, Team2: 80, Team3: 80 },
  ];

  return (
    <Card className="bg-white/80 backdrop-blur-sm border border-slate-
200 shadow-sm">
      <CardHeader>
        <CardTitle>Team Skill Distribution</CardTitle>
        <CardDescription>
          Comparing skills across teams
        </CardDescription>
      </CardHeader>
      <CardContent>
        <BaseRadarChart
          data={data}
          dataKeys={['Team1', 'Team2', 'Team3']}
          labelKey="skill"
          height={350}
          domain={[0, 100]}
        />
      </CardContent>
    </Card>
  );
}
```

Example 3: Performance Distribution

```
import { Card, CardHeader, CardTitle, CardDescription, CardContent }
from "@/components/ui/core/Card";
import { BasePieChart } from "@/components/ui/charts/core/BasePieChart";

export function PerformanceDistribution() {
  const data = [
    { name: "Star", value: 2 },
    { name: "Strong", value: 3 },
    { name: "Solid", value: 3 },
    { name: "Lower", value: 1 },
    { name: "Poor", value: 1 },
```

```
  ];

  return (
    <Card className="bg-white/80 backdrop-blur-sm border border-slate-
200 shadow-sm">
      <CardHeader>
        <CardTitle>Performance Distribution</CardTitle>
        <CardDescription>
          Team members across performance categories
        </CardDescription>
      </CardHeader>
      <CardContent>
        <BasePieChart
          data={data}
          innerRadius={50}
          outerRadius={90}
          paddingAngle={2}
          labelType="name-percent"
          height={300}
        />
      </CardContent>
    </Card>
  );
}
```

Example 4: Revenue vs Costs Area Chart

```
import { Card, CardHeader, CardTitle, CardDescription, CardContent }
from "@/components/ui/core/Card";
import { BaseAreaChart } from
"@/components/ui/charts/core/BaseAreaChart";

export function RevenueVsCostsChart() {
  const data = [
    { month: 'Jan', revenue: 12000, costs: 8000 },
    { month: 'Feb', revenue: 15000, costs: 9000 },
    { month: 'Mar', revenue: 18000, costs: 8500 },
    { month: 'Apr', revenue: 14000, costs: 9500 },
    { month: 'May', revenue: 21000, costs: 10000 },
    { month: 'Jun', revenue: 22000, costs: 10500 },
  ];

  return (
    <Card className="bg-white/80 backdrop-blur-sm border border-slate-
200 shadow-sm">
      <CardHeader>
        <CardTitle>Revenue vs Costs</CardTitle>
        <CardDescription>
          Monthly revenue and costs comparison
        </CardDescription>
```

```
        </CardHeader>
        <CardContent>
          <BaseAreaChart
            data={data}
            xKey="month"
            yKeys={["revenue", "costs"]}
            height={300}
            stacked={false}
            curved={true}
            showDots={true}
            variant="sequential"
          />
        </CardContent>
      </Card>
    );
  }
```

This documentation covers the core functionality and implementation details of our chart system. For specific questions or advanced use cases, please consult the front-end team or refer to the Recharts documentation at recharts.org.