Chart.js for React Your Crypto Chart Buddy Guide

Grok, Your Friendly AI Coding Assistant

June 17, 2025

A Beginner-Friendly Guide to Building a Cryptocurrency Price Chart with React and Chart.js

Contents

1 Introduction: Lets Build a Cool Crypto Chart!

Welcome to your **Chart.js for React Buddy Guide**! This guide is designed to help you, even if youre just starting out, build a **line chart** that shows cryptocurrency prices (like Bitcoin or Ethereum) over different time ranges (1 day, 7 days, 30 days, or 1 year). Well use **React** for the app, **Chart.js** for the chart, and the **CoinGecko API** to get real price data. Lets make it fun and simple!

1.1 What Youll Learn

By the end of this guide, youll know how to:

- Fetch real-time crypto price data from the CoinGecko API.
- Create a smooth line chart using Chart.js and react-chartjs-2.
- Let users switch between different time ranges (1D, 7D, 30D, 1Y).
- Style your chart to look professional.
- Troubleshoot common issues like a pro.

1.2 Who Is This For?

This guide is perfect for:

- Beginners learning React or Chart.js.
- Anyone curious about building data visualizations.
- Crypto enthusiasts who want to track prices in a custom app.

2 Getting Started: Setting Up Your Project

Before we write code, lets set up our project. Think of this as gathering your tools before building a toy rocket!

2.1 Create a React App

If you dont have a React app yet, create one using Create React App:

```
npx create-react-app crypto-chart
cd crypto-chart
```

Listing 1: Creating a new React app

2.2 Install Dependencies

We need two packages: `chart.js` (the chart engine) and `react-chartjs-2` (a React wrapper for Chart.js). Install them with:

```
npm install chart.js react-chartjs-2
```

Listing 2: Installing Chart.js and react-chartjs-2

2.3 Check Your Setup

Start your app to make sure everything works:

```
npm start
```

Listing 3: Starting the React app

Open http://localhost:3000 in your browser. You should see the default React welcome page.

3 Building the Chart: The Code Explained

Now, lets write the code for our `CryptoChart` component. Well break it down into small, bite-sized pieces so its easy to understand.

3.1 The Full Code

Heres the complete code for your `CryptoChart` component. Dont worry if it looks longwell explain each part next!

```
import React, { useEffect, useState } from 'react';
import { Line } from 'react-chartjs-2';
import {
  Chart as ChartJS,
  LineElement,
  PointElement,
  CategoryScale,
  LinearScale,
  Tooltip,
  Legend,
} from 'chart.js';

// Register Chart.js components
ChartJS.register(LineElement, PointElement, CategoryScale,
   LinearScale, Tooltip, Legend);

// Time range options
const timeOptions = [
  { label: '1D', days: 1 },
  { label: '7D', days: 7 },
  { label: '30D', days: 30 },
  { label: '1Y', days: 365 },
];

const CryptoChart = ({ coinId }) => {
  const [chartData, setChartData] = useState(null);
  const [days, setDays] = useState(7);
  const [loading, setLoading] = useState(false);

  useEffect(() => {
    const fetchData = async () => {
      setLoading(true);
      try {
        const res = await fetch(
          `https://api.coingecko.com/api/v3/coins/${coinId}/market_chart?vs_
        );
        const data = await res.json();

        // Format dates for labels
        const labels = data?.prices?.map(p =>
          new Date(p[0]).toLocaleDateString('en-US', {
            month: 'short',
            year: days >= 180 ? 'numeric' : undefined,
            day: days < 180 ? 'numeric' : undefined,
          })
        );

        // Extract prices
        const prices = data?.prices?.map(p => p[1]);

        // Set chart data
        setChartData({
          labels,
          datasets: [
            {
              label: `${coinId.toUpperCase()} Price (USD)`,
              data: prices,
              borderColor: '#3b82f6',
              backgroundColor: 'rgba(59, 130, 246, 0.2)',
              fill: true,
              tension: 0.3,
            },
          ],
        });
      } catch (error) {
        console.error('Error fetching data:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, [coinId, days]);

  return (
    <div className="w-full p-4">
      <h2 className="text-2xl font-bold
         mb-4">{coinId.toUpperCase()} Price Chart</h2>
      {loading ? (
        <p className="text-center">Loading chart...</p>
      ) : (
        chartData && <Line data={chartData} />
      )}
      <div className="flex gap-2 mt-4 justify-center">
        {timeOptions.map((opt) => (
          <button
            key={opt.days}
            onClick={() => setDays(opt.days)}
            className={`px-3 py-1 rounded text-sm ${
              days === opt.days ? 'bg-blue-600 text-white' :
                 'bg-gray-300'
            }`}
          >
            {opt.label}
          </button>
        ))}
      </div>
    </div>
  );
};

export default CryptoChart;
```

Listing 4: CryptoChart.jsx

3.2 How to Use It

In your `App.jsx`, import and use the component like this:

```
import CryptoChart from './CryptoChart';

function App() {
  return (
    <div className="App">
      <CryptoChart coinId="bitcoin" />
    </div>
  );
}

export default App;
```

Listing 5: App.jsx

Replace `bitcoin` with any CoinGecko coin ID (e.g., `ethereum`, `dogecoin`).

3.3 Code Breakdown

Lets go through the code step-by-step, like exploring a treasure map!

3.3.1 Imports

```
import React, { useEffect, useState } from 'react';
import { Line } from 'react-chartjs-2';
import { Chart as ChartJS, LineElement, PointElement,
    CategoryScale, LinearScale, Tooltip, Legend } from 'chart.js';
```

- `useEffect` and `useState`: React hooks to manage data and side effects.
- `Line`: The line chart component from `react-chartjs-2`.
- `ChartJS components`: Pieces needed to build the chart (like lines, points, and scales).

3.3.2 Registering Chart.js Components

```
ChartJS.register(LineElement, PointElement, CategoryScale,
    LinearScale, Tooltip, Legend);
```

This line is like telling Chart.js, Heres the toolbox I need to build my line chart! Chart.js is modular, so you only register the components you need to keep your app lightweight. Lets break down each component and why its essential for your cryptocurrency price chart:

- **LineElement**: Draws the line connecting your data points, showing the trend of crypto prices (e.g., Bitcoins price over time). Without it, youd have no linejust floating dots or nothing at all! In your chart, it uses the `borderColor` and `tension` (0.3 for a smooth curve) to render a blue line.
- **PointElement**: Renders the individual data points (dots) on the line, representing each price at a specific time (e.g., $35,000 on Jun 16). These dots are interactive, showing up in tooltips when hovered. You can customize them with `pointRadius` or `pointBackgroundColor`.
- **CategoryScale**: Manages the x-axis, treating your dates (e.g., Jun 16, Jun 17) as categories. It spaces them evenly, making the chart easy to read, even if the data points arent perfectly spaced in time.
- **LinearScale**: Handles the y-axis, scaling numerical prices (e.g., $30,000 to $40,000). It automatically calculates tick marks so users can see price changes clearly.
- **Tooltip**: Shows a popup when you hover over a data point, displaying the date and price (e.g., Jun 16, 2025: $35,000). It makes your chart interactive and user-friendly.
- **Legend**: Displays a legend (e.g., Bitcoin Price (USD)) to explain what the line represents. Its especially useful if you add multiple coins later.

Why Register? Chart.js requires explicit registration to include only the features you need, reducing bundle size and enabling tree-shaking (removing unused code). If you forget a component, your chart might not render or throw errors like Chart is not defined.

Example Customization: Add a title to the axes and style the legend:

```
const options = {
  plugins: {
    legend: { position: 'top', labels: { font: { size: 14 } } },
  },
  scales: {
    x: { title: { display: true, text: 'Date' } },
    y: { title: { display: true, text: 'Price (USD)' } },
  },
};
```

Use it in your render: `<Line data={chartData} options={options} />`.

3.3.3 Additional Chart.js Components

Want to take your chart to the next level? Here are some extra components and plugins you can use:

- **TimeScale**: Replaces `CategoryScale` for the x-axis to handle real dates and times, spacing data points based on actual time intervals. This is great for crypto data, as CoinGecko returns timestamps. To use it, install `chartjs-adapter-date-fns`:

  ```
  npm install date-fns chartjs-adapter-date-fns
  ```

  Then register and configure:

  ```
  import { TimeScale } from 'chart.js';
  import 'chartjs-adapter-date-fns';
  ChartJS.register(TimeScale, LineElement, PointElement,
     LinearScale, Tooltip, Legend);

  const labels = data?.prices?.map(p => p[0]); // Use raw
     timestamps
  setChartData({
    labels,
    datasets: [
      {
        label: `${coinId.toUpperCase()} Price (USD)`,
        data: prices,
        borderColor: '#3b82f6',
        backgroundColor: 'rgba(59, 130, 246, 0.2)',
        fill: true,
        tension: 0.3,
      },
    ],
  });
  const options = {
    scales: {
      x: {
        type: 'time',
        time: { unit: days <= 1 ? 'hour' : 'day' },
        title: { display: true, text: 'Date' },
      },
    },
  };
  ```

- **Filler**: Enables the `fill` property to shade the area under the line. Youre already using it (`fill: true`), but you can register `Filler` explicitly for advanced fills like gradients.
- **Annotation Plugin**: Add lines or labels to highlight key price points (e.g., $40,000). Install `chartjs-plugin-annotation` and configure:

```
import annotationPlugin from 'chartjs-plugin-annotation';
ChartJS.register(annotationPlugin, LineElement,
   PointElement, CategoryScale, LinearScale, Tooltip,
   Legend);
const options = {
  plugins: {
    annotation: {
      annotations: {
        line1: {
          type: 'line',
          yMin: 40000,
          yMax: 40000,
          borderColor: 'red',
          borderWidth: 2,
          label: { content: '$40,000 Milestone', enabled:
             true },
        },
      },
    },
  },
};
```

3.3.4 Time Range Options

```
const timeOptions = [
  { label: '1D', days: 1 },
  { label: '7D', days: 7 },
  { label: '30D', days: 30 },
  { label: '1Y', days: 365 },
];
```

These are the buttons users will click to switch between time ranges.

3.3.5 State Management

```
const [chartData, setChartData] = useState(null);
const [days, setDays] = useState(7);
const [loading, setLoading] = useState(false);
```

- `chartData`: Holds the charts labels and prices.
- `days`: Tracks the selected time range (default is 7 days).
- `loading`: Shows a loading message while fetching data.

3.3.6 Fetching Data

```
useEffect(() => {
  const fetchData = async () => {
    setLoading(true);
    try {
      const res = await fetch(
        `https://api.coingecko.com/api/v3/coins/${coinId}/market_chart?vs_cu
      );
      const data = await res.json();
      // ... (data processing)
    } catch (error) {
      console.error('Error fetching data:', error);
    } finally {
      setLoading(false);
    }
  };
  fetchData();
}, [coinId, days]);
```

This code runs when `coinId` or `days` changes, fetching price data from CoinGecko.

3.3.7 Formatting Data

```
const labels = data?.prices?.map(p =>
  new Date(p[0]).toLocaleDateString('en-US', {
    month: 'short',
    year: days >= 180 ? 'numeric' : undefined,
    day: days < 180 ? 'numeric' : undefined,
  })
);
const prices = data?.prices?.map(p => p[1]);
```

Converts timestamps to readable dates and extracts prices.

3.3.8 Setting Chart Data

```
setChartData({
  labels,
  datasets: [
    {
      label: `${coinId.toUpperCase()} Price (USD)`,
      data: prices,
      borderColor: '#3b82f6',
      backgroundColor: 'rgba(59, 130, 246, 0.2)',
      fill: true,
      tension: 0.3,
    },
  ],
});
```

Tells Chart.js how to draw the chart with labels and prices.

3.3.9 Rendering the Chart

```
{loading ? (
  <p className="text-center">Loading chart...</p>
) : (
  chartData && <Line data={chartData} />
)}
```

Shows a loading message or the chart.

3.3.10 Time Range Buttons

```
<div className="flex gap-2 mt-4 justify-center">
  {timeOptions.map((opt) => (
    <button
      key={opt.days}
      onClick={() => setDays(opt.days)}
      className={`px-3 py-1 rounded text-sm ${
        days === opt.days ? 'bg-blue-600 text-white' :
           'bg-gray-300'
      }`}
    >
      {opt.label}
    </button>
  ))}
</div>
```

Renders buttons for time range selection.

4 Customizing Your Chart

Want to make your chart even cooler? Here are some ideas!

4.1 Add Chart Options

Customize the charts appearance:

```
const options = {
  responsive: true,
  plugins: {
    legend: { position: 'top' },
    tooltip: { mode: 'index', intersect: false },
  },
  scales: {
    x: { title: { display: true, text: 'Date' } },
    y: { title: { display: true, text: 'Price (USD)' } },
  },
};
```

Use it: `<Line data={chartData} options={options} />`.

4.2 Change Colors

Try different colors:

```
borderColor: '#10b981', // Emerald green
backgroundColor: 'rgba(16, 185, 129, 0.2)', // Light green fill
```

4.3 Add More Time Ranges

Add a 3-month option:

```
const timeOptions = [
  { label: '1D', days: 1 },
  { label: '7D', days: 7 },
  { label: '30D', days: 30 },
  { label: '90D', days: 90 },
  { label: '1Y', days: 365 },
];
```

5 Troubleshooting: Fixing Common Issues

Table 1: Common Issues and Fixes

Problem	**What It Means**	**How to Fix**
Cannot read property 'map'	Data didnt load properly	Add checks: `if (data?.prices)`
Blank chart	`chartData` is null	Wrap with `chartData && <Line />`
Ugly dates	Timestamps not formatted	Use `toLocaleDateString`
API error	CoinGecko rate limit or bad `coinId`	Check `coinId`, add error handling
Chart not rendering	Missing component	Ensure all components are registered

6 Next Steps: Level Up Your Chart

- **Multiple Coins**: Add a dropdown to switch between coins.
- **Volume Data**: Show trading volume as a second dataset.
- **Tailwind CSS**: Style with Tailwind for a modern look.
- **Error Messages**: Show user-friendly error messages.

7 Conclusion

Youve built a cryptocurrency price chart with React and Chart.js! You learned how to fetch data, use Chart.js components, and make your chart interactive. Keep experimenting, and Im here to help!