App.tsx   
  
import React, { useEffect, useRef } from 'react';

import styled, { createGlobalStyle } from 'styled-components';

import { Chart, CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend } from 'chart.js/auto';

Chart.register(CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend);

const GlobalStyle = createGlobalStyle`

/\* Global styles for responsive tables \*/

table {

width: 100%;

border-collapse: collapse;

}

th, td {

padding: 8px;

text-align: left;

border-bottom: 1px solid #ddd;

}

@media (max-width: 768px) {

th, td {

display: block;

text-align: right;

}

th {

position: absolute;

left: -9999px;

top: -9999px;

}

tr {

border: 1px solid #ccc;

margin-bottom: 10px;

}

td {

border: none;

position: relative;

padding-left: 50%;

text-align: left;

}

td:before {

content: attr(data-label);

position: absolute;

left: 0;

width: 50%;

padding-left: 10px;

font-weight: bold;

text-align: left;

}

}

`;

const ChartContainer = styled.div<{ width: string; height: string }>`

background-color: #2a2d3e;

padding: 20px;

border-radius: 12px;

box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);

color: white;

width: ${({ width }) => width};

height: ${({ height }) => height};

margin: 20px;

@media (max-width: 1200px) {

width: 90%;

height: 500px;

}

@media (max-width: 768px) {

width: 95%;

height: 400px;

}

@media (max-width: 480px) {

width: 100%;

height: 300px;

}

`;

const TitleStyled = styled.h3`

margin-bottom: 0px;

text-align: center;

`;

const LoadingMessage = styled.div`

text-align: center;

margin: 20px 0;

`;

interface ChartComponentProps {

coinData: any[];

width?: string;

height?: string;

}

const ChartComponent: React.FC<ChartComponentProps> = ({ coinData, width = '95%', height = '450px' }) => {

const chartRef = useRef<HTMLCanvasElement | null>(null);

useEffect(() => {

if (!Array.isArray(coinData) || coinData.length === 0 || !coinData[0]?.sparkline\_in\_7d?.price) {

return;

}

const ctx = chartRef.current?.getContext('2d');

if (!ctx) return;

const labels = coinData[0].sparkline\_in\_7d.price.map((\_: number, index: number) =>

new Date(Date.now() - (6 - index) \* 24 \* 60 \* 60 \* 1000).toLocaleDateString('en-US', { month: 'short', year: 'numeric' })

);

const chart = new Chart(ctx, {

type: 'line',

data: {

labels,

datasets: coinData.map((coin: any) => ({

label: `${coin.name} Price`,

data: coin.sparkline\_in\_7d.price,

borderColor: getRandomColor(),

backgroundColor: getRandomColor(0.2),

tension: 0.1,

fill: false,

pointRadius: 3,

pointHoverRadius: 6,

})),

},

options: {

responsive: true,

maintainAspectRatio: false,

plugins: {

legend: {

display: true,

position: 'top',

labels: {

color: 'white',

font: {

size: 14,

},

},

},

tooltip: {

mode: 'index',

intersect: false,

},

},

scales: {

x: {

display: true,

title: {

display: true,

text: 'Date',

color: '#ffffff',

font: {

size: 16,

},

},

ticks: {

color: '#ffffff',

font: {

size: 12,

family: 'Arial',

weight: 'bold',

},

callback: function (value: number | string, index: number, values: any) {

const label = this.getLabelForValue(value as number).toString();

return label.split(',')[0];

},

},

grid: {

color: 'rgba(255, 255, 255, 0.1)',

},

},

y: {

display: true,

title: {

display: true,

text: 'Price (USD)',

color: '#ffffff',

font: {

size: 16,

},

},

ticks: {

color: '#ffffff',

font: {

size: 12,

},

},

grid: {

color: 'rgba(255, 255, 255, 0.1)',

},

},

},

},

});

return () => {

chart.destroy();

};

}, [coinData]);

if (!Array.isArray(coinData) || coinData.length === 0 || !coinData[0]?.sparkline\_in\_7d?.price) {

return <LoadingMessage>Loading chart...</LoadingMessage>;

}

function getRandomColor(opacity = 1) {

const randomColor = `rgba(${Math.floor(Math.random() \* 255)}, ${Math.floor(Math.random() \* 255)}, ${Math.floor(Math.random() \* 255)}, ${opacity})`;

return randomColor;

}

return (

<>

<GlobalStyle />

<ChartContainer width={width} height={height}>

<TitleStyled>7 Day Price Trend</TitleStyled>

<canvas ref={chartRef}></canvas>

</ChartContainer>

</>

);

};

export default ChartComponent;