# Guía mejorada: Integración de ApexCharts en NEXA (Ionic + React)

Versión ampliada y visualmente mejorada. Incluye ejemplos completos de código, sugerencias de estilo, accesibilidad y pasos para exportar a PDF.

## 1. Resumen y objetivo

Esta guía explica paso a paso cómo integrar gráficos interactivos con **ApexCharts** en una aplicación **Ionic + React** llamada *NEXA*, consumiendo datos desde **Supabase** (o un endpoint FastAPI). Se enfatiza en:

* Código listo para copiar/pegar (TypeScript + React).
* Buenas prácticas para rendimiento y accesibilidad.
* Sugerencias visuales (contenedores, layout, animaciones suaves).
* Actualización en tiempo real con **Supabase Realtime**.

## 2. Requisitos previos

* Node.js ≥ 16
* npm o yarn
* Un proyecto Ionic React (creado con ionic start o similar)
* Cuenta y tabla en Supabase con datos de movimientos / stock

## 3. Instalación de dependencias

Desde la carpeta frontend/ del proyecto ejecuta:

npm install apexcharts react-apexcharts @supabase/supabase-js  
# o con yarn  
# yarn add apexcharts react-apexcharts @supabase/supabase-js

Si usas TypeScript, instala tipos si lo necesitas (la librería suele incluirlos):

npm install --save-dev @types/apexcharts

## 4. Variables de entorno

Crea frontend/.env.local con las variables públicas de Supabase (no guardes la SERVICE\_ROLE en frontend):

REACT\_APP\_SUPABASE\_URL=https://xxxxx.supabase.co  
REACT\_APP\_SUPABASE\_KEY=eyJhbGciOiJI...

En TypeScript/React usa process.env.REACT\_APP\_SUPABASE\_URL etc. Asegúrate de reiniciar el servidor tras cambiar .env.local.

## 5. Esquema de datos y función RPC (ejemplo)

Suponiendo una tabla movimientos con columnas relevantes:

* id (bigint)
* product\_id (bigint)
* cantidad (integer)
* tipo (text) — e.g. ‘entrada’ / ‘salida’
* created\_at (timestamp)

### Función SQL para agregar stock por día (Supabase SQL)

create or replace function public.stock\_by\_day(p\_product\_id bigint default null)  
returns table(day date, stock integer)  
language sql  
as $$  
select  
 date(created\_at) as day,  
 sum(case when tipo = 'entrada' then cantidad  
 when tipo = 'salida' then -cantidad  
 else 0 end) as stock  
from movimientos  
where (p\_product\_id is null or product\_id = p\_product\_id)  
group by date(created\_at)  
order by day;  
$$;

Nota: Ajusta los nombres de tabla/columnas a tu esquema real. Esta función devuelve la evolución diaria del stock (puedes transformar a acumulado en el frontend si lo deseas).

## 6. Inicializar cliente Supabase (src/lib/supabaseClient.ts)

import { createClient } from '@supabase/supabase-js';  
  
const url = process.env.REACT\_APP\_SUPABASE\_URL!;  
const key = process.env.REACT\_APP\_SUPABASE\_KEY!;  
  
export const supabase = createClient(url, key);

## 7. Componente completo: src/components/StockChart.tsx

Archivo TypeScript con uso de react-apexcharts, hooks, fetch a la RPC y suscripción en tiempo real.

import React, { useEffect, useMemo, useState } from 'react';  
import dynamic from 'next/dynamic'; // si usas Next.js, sino importa normalmente  
import Chart from 'react-apexcharts';  
import { supabase } from '../lib/supabaseClient';  
  
type Point = { day: string; stock: number };  
  
export interface StockChartProps {  
 productId?: number | null;  
 height?: number;  
}  
  
export const StockChart: React.FC<StockChartProps> = ({ productId = null, height = 320 }) => {  
 const [data, setData] = useState<Point[]>([]);  
 const [loading, setLoading] = useState(true);  
  
 useEffect(() => {  
 let isMounted = true;  
 setLoading(true);  
  
 async function load() {  
 // Llamada a la función RPC  
 const { data: rows, error } = await supabase.rpc('stock\_by\_day', { p\_product\_id: productId });  
 if (error) {  
 console.error('RPC error', error);  
 setLoading(false);  
 return;  
 }  
 if (!isMounted) return;  
 setData((rows as any[]).map(r => ({ day: r.day, stock: Number(r.stock) })) || []);  
 setLoading(false);  
 }  
  
 load();  
  
 // Suscripción Realtime - escucha inserciones en movimientos y refresca  
 const subscription = supabase  
 .channel('public:movimientos')  
 .on('postgres\_changes', { event: '\*', schema: 'public', table: 'movimientos' }, payload => {  
 // Para simplicidad: recargar la RPC completa.  
 // Para optimizar: aplicar patch incremental al estado.  
 load();  
 })  
 .subscribe();  
  
 return () => {  
 isMounted = false;  
 supabase.removeChannel(subscription);  
 };  
 }, [productId]);  
  
 const series = useMemo(() => [  
 {  
 name: 'Stock',  
 data: data.map(d => ({ x: d.day, y: d.stock }))  
 }  
 ], [data]);  
  
 const options = useMemo(() => ({  
 chart: {  
 id: 'stock-area',  
 zoom: { enabled: true },  
 toolbar: { show: true },  
 animations: { enabled: true }  
 },  
 stroke: { curve: 'smooth' },  
 xaxis: {  
 type: 'datetime',  
 labels: { datetimeUTC: false }  
 },  
 yaxis: { title: { text: 'Unidades' } },  
 tooltip: { x: { format: 'dd MMM yyyy' } },  
 dataLabels: { enabled: false },  
 markers: { size: 3 }  
 }), []);  
  
 return (  
 <div className="p-4 bg-white rounded-2xl shadow-sm dark:bg-gray-800">  
 <div className="flex items-center justify-between mb-2">  
 <h3 className="text-lg font-semibold">Evolución de stock</h3>  
 {loading && <small className="text-sm text-gray-500">Cargando…</small>}  
 </div>  
  
 <div style={{ height }}>  
 <Chart options={options as any} series={series as any} type="area" height={height} />  
 </div>  
  
 <div className="mt-3 text-xs text-gray-500">Actualizado en tiempo real vía Supabase Realtime</div>  
 </div>  
 );  
};  
  
export default StockChart;

Tips: - Para optimizar la suscripción, filtra por product\_id en el channel o aplica un patch incremental en lugar de recargar todo. - Si trabajas con conjuntos de datos grandes, haz agregación en el servidor (SQL) y trae solo lo necesario.

## 8. Uso en una página de Ionic (Dashboard)

Ejemplo: src/pages/Dashboard.tsx (Ionic React)

import React from 'react';  
import { IonPage, IonHeader, IonToolbar, IonTitle, IonContent } from '@ionic/react';  
import StockChart from '../components/StockChart';  
  
const Dashboard: React.FC = () => {  
 return (  
 <IonPage>  
 <IonHeader>  
 <IonToolbar>  
 <IonTitle>Dashboard</IonTitle>  
 </IonToolbar>  
 </IonHeader>  
  
 <IonContent className="p-4">  
 <div className="grid gap-4 grid-cols-1 md:grid-cols-2">  
 <StockChart productId={null} height={320} />  
 {/\* Otros widgets \*/}  
 </div>  
 </IonContent>  
 </IonPage>  
 );  
};  
  
export default Dashboard;

## 9. Sugerencias visuales y UX

* Contenedor: usa tarjetas con rounded-2xl, sombra suave y padding amplio para móviles.
* Colores: integra el tema de tu app; ApexCharts acepta theme: { mode: 'dark'|'light' }.
* Animaciones: evita animaciones pesadas en datasets grandes.
* Interacción: habilita zoom y pan, y un botón para restablecer el zoom.
* Leyendas y filtros: añade controles sobre la serie (select para producto/periodo).

CSS de ejemplo (Tailwind útil en Ionic + React si ya lo usas):

/\* tailwind utilities example \*/  
.card { @apply bg-white rounded-2xl p-4 shadow-md }

## 10. Accesibilidad

* Asegura aria-labels apropiados para controles.
* Provee una tabla alternativa (HTML) con datos base y descarga CSV para usuarios que no pueden interactuar con gráficos.
* Contraste: verifica contraste de colores en tooltips/etiquetas.

## 11. Performance y escala

* Agrega paginación/limit en consultas históricas.
* Mantén agregaciones en SQL para reducir payload.
* Usa requestAnimationFrame sólo si ejecutas animaciones manuales.

## 12. Pruebas y troubleshooting rápido

* Si el chart queda en blanco: revisa que series no esté vacío y que xaxis.type coincida con el formato de fechas.
* Errores de CORS o key inválida: verifica REACT\_APP\_SUPABASE\_URL y KEY.
* Realtime no se conecta: confirma que la tabla esté configurada para Realtime en Supabase.

## 13. Exportar como PDF / Imágenes

* En el editor del documento (canvas) puedes exportar a PDF.
* Para capturas de alta calidad de gráficos, usa svg/png export de ApexCharts (chart.dataURI() o chart.export() según la API).

## 14. Código adicional útil

### 14.1: Función para convertir resultados a serie acumulada

function acumulado(points: { day: string; stock: number }[]) {  
 const res: { day: string; stock: number }[] = [];  
 let total = 0;  
 for (const p of points) {  
 total += p.stock;  
 res.push({ day: p.day, stock: total });  
 }  
 return res;  
}

### 14.2: Export CSV simple

function exportCSV(rows: any[], filename = 'datos.csv'){  
 const csv = [Object.keys(rows[0]).join(','), ...rows.map(r => Object.values(r).join(','))].join('\n');  
 const blob = new Blob([csv], { type: 'text/csv' });  
 const url = URL.createObjectURL(blob);  
 const a = document.createElement('a');  
 a.href = url; a.download = filename; a.click();  
 URL.revokeObjectURL(url);  
}

## 15. Resumen y próximos pasos

1. Copia los archivos sugeridos al proyecto y ajusta nombres de tablas/columnas.
2. Testea la RPC en Supabase SQL editor.
3. Integra el componente en Dashboard y valida en móvil.
4. Mejora visual agregando iconografía y controles de periodo.

## Anexos

* Checklist de deploy en producción (variables, CORS, límites de supabase).
* Posibles mejoras: usar cache SWR/React Query para manejo de caché y revalidación.

### ¿Qué más quieres incluir?

* ¿Quieres que además genere una versión PPTX con estas diapositivas resumidas?
* ¿Prefieres que exporte directamente a PDF listo para presentación?

*Documento generado automáticamente — si quieres cambios de estilo (tema oscuro, paleta corporativa, imágenes), dime y lo adapto.*