Project: performance-dashboard (Next.js 14 + TypeScript) This codebase implements a high-performance real-time dashboard built with Next.js 14 App Router + TypeScript. It includes a canvas-based LineChart, simulated real-time data arriving every 100ms, FPS & memory monitor, basic controls, data aggregation, virtualized data table and an API route to stream/generate data. --- ## File tree (created files in this single document) performance-dashboard/ ├— app/ | ├— dashboard/ | | ├— page.tsx | | └— layout.tsx | ├— api/ | | └— data/ | | └— route.ts | ├— globals.css | └— layout.tsx ├— components/ | ├— charts/ | | └— LineChart.tsx | ├— controls/ | | └— TimeRangeSelector.tsx | ├— ui/ | | ├— DataTable.tsx | | └— PerformanceMonitor.tsx | └— providers/ | └— DataProvider.tsx ├— hooks/ | ├— useDataStream.ts | └— usePerformanceMonitor.ts ├— lib/ | ├— dataGenerator.ts | └— types.ts ├— package.json ├— next.config.js ├— tsconfig.json ├— README.md └— PERFORMANCE.md --- ## Important: How to run (summary) 1. Save the project files into a folder named performance-dashboard. 2. Run npm install. 3. Run npm run dev (Next.js dev server). 4. Open http://localhost:3000/dashboard to view the dashboard. --- ## Key code files (full contents) --- app/layout.tsx

React TSX□□

```
import './globals.css';

export const metadata = { title: 'Performance Dashboard' };

export default function RootLayout({ children }: { children: React.ReactNode }) {
 return (
  <html lang="en">
   <body>
    <main>{children}</main>
   </body>
  </html>
 );
}
```

--- app/dashboard/layout.tsx

React TSX□□

```
export default function DashboardLayout({ children }: { children: React.ReactNode }) {
 return <div style={{padding:20, fontFamily:'Inter, system-ui, sans-serif'}}>{children}</div>;
}
```

--- app/dashboard/page.tsx

React TSX□□

```
import DataProvider from '../../components/providers/DataProvider';
import dynamic from 'next/dynamic';
```

```
import PerformanceMonitor from '../../components/ui/PerformanceMonitor';
import TimeRangeSelector from '../../components/controls/TimeRangeSelector';

const LineChart = dynamic(() => import('../../components/charts/LineChart'), { ssr: false });

export default function Page() {
  return (
    <DataProvider>
      <h1>Performance Dashboard — Next.js + TypeScript</h1>
      <div style={{display:'grid', gridTemplateColumns:'1fr 320px', gap:20}}>
        <div style={{background:'#fff', padding:10, borderRadius:8}}>
          <LineChart />
        </div>
        <aside style={{display:'flex', flexDirection:'column', gap:12}}>
          <PerformanceMonitor />
          <TimeRangeSelector />
        </aside>
      </div>
    </DataProvider>
  );
}
--- app/api/data/route.ts
TypeScript□□
import { NextResponse } from 'next/server';
import { generateDataPoint } from '../../../lib/dataGenerator';

export async function GET(request: Request) {
  // Simple endpoint returning an initial batch — used by server component if desired
  const points = Array.from({length: 10000}).map((_,i)=> generateDataPoint(Date.now() - (10000-
i)*100, i%5));
  return NextResponse.json({ points });
```

}

--- lib/types.ts

TypeScript

export interface DataPoint { timestamp: number; value: number; category?: string; }

--- lib/dataGenerator.ts

TypeScript

import { DataPoint } from './types';

export function generateDataPoint(timestamp = Date.now(), seed = 0): DataPoint {

// simple pseudo-random walk

const base = Math.sin((timestamp/1000) + seed) * 20 + 50;

const noise = (Math.random()-0.5)*5;

return { timestamp, value: Math.max(0, base + noise), category: `c${seed%5}` };

}

--- components/providers/DataProvider.tsx `tsx 'use client'; import React, { createContext, useContext, useEffect, useRef, useState } from 'react'; import { DataPoint } from '../../lib/types'; import { generateDataPoint } from '../../lib/dataGenerator'; type T = { data: DataPoint[]; append: (p:DataPoint)=>void; }; const DataContext = createContext<T | null>(null);

export default function DataProvider({ children }: { children: React.ReactNode }) { const [data, setData] = useState<DataPoint[]>(() => { // initial window (10k points) const now = Date.now(); return Array.from({length:10000}).map((_,i)=> generateDataPoint(now - (10000-i)*100, i%5)); }); const append = (p:DataPoint)=> setData(prev=> { const next = prev.slice(-20000); // keep sliding window next.push(p); return next; }); // simulated stream every 100ms useEffect(()=>{ let mounted = true; const t = setInterval(()=>{ if (!mounted) return; append(generateDataPoint()); }, 100); return ()=>{ mounted=false; clearInterval(t); }; },[]); return <DataContext.Provider value={{data, append}}>{children}</DataContext.Provider>; } export function useDataContext(){ const v = useContext(DataContext); if (!v) throw new Error('useDataContext must be used inside DataProvider'); return v; }

copy

--- hooks/useDataStream.ts

ts // lightweight hook to expose data to components import { useEffect, useMemo, useState } from 'react'; import { useDataContext } from '../components/providers/DataProvider'; import { DataPoint } from '../lib/types'; export function useDataWindow(msWindow = 60_000) { const { data } = useDataContext(); const windowData = useMemo(()=> { const cutoff = Date.now() - msWindow; return data.filter(d=> d.timestamp >= cutoff); }, [data, msWindow]); return windowData as DataPoint[]; }

copy

--- hooks/usePerformanceMonitor.ts

ts 'use client'; import { useEffect, useRef, useState } from 'react'; export function useFPS() { const last = useRef<number | null>(null); const frames = useRef(0); const lastTick = useRef(performance.now()); const [fps, setFps] = useState(0); useEffect(()=>{ let mounted = true; function tick(now:number){ if (!mounted) return; frames.current++; if (now - lastTick.current >= 500) { const f = Math.round((frames.current * 1000) / (now - lastTick.current)); setFps(f); frames.current = 0; lastTick.current = now; } requestAnimationFrame(tick); } const id = requestAnimationFrame(tick); return ()=>{ mounted=false; cancelAnimationFrame(id); }; },[]); return fps; }

copy

--- components/charts/LineChart.tsx

tsx 'use client'; import React, { useEffect, useRef } from 'react'; import { useDataWindow } from '../../hooks/useDataStream'; import { useFPS } from '../../hooks/usePerformanceMonitor'; export default function LineChart(){ const canvasRef = useRef<HTMLCanvasElement | null>(null); const data = useDataWindow(60000); const fps = useFPS(); useEffect(()=>{ const canvas = canvasRef.current!; const ctx = canvas.getContext('2d')!; let raf = 0; const DPR = Math.max(1, window.devicePixelRatio || 1); function resize(){ const w = canvas.clientWidth; const h = canvas.clientHeight; canvas.width = Math.round(w*DPR); canvas.height = Math.round(h*DPR); ctx.setTransform(DPR,0,0,DPR,0,0); } resize(); window.addEventListener('resize', resize); function render(){ const w = canvas.clientWidth; const h = canvas.clientHeight; ctx.clearRect(0,0,w,h); // background ctx.fillStyle = '#0f172a'; ctx.fillRect(0,0,w,h); // draw axis ctx.strokeStyle = '#334155'; ctx.lineWidth = 1; ctx.beginPath(); ctx.moveTo(40,10); ctx.lineTo(40,h-30); ctx.lineTo(w-10,h-30); ctx.stroke(); if (data.length>1) { // map timestamps to x const t0 = data[0].timestamp; const t1 = data[data.length-1].timestamp; const minV = Math.min(...data.map(d=>d.value)); const maxV = Math.max(...data.map(d=>d.value)); const marginL = 40; const marginR = 10; const marginT=10; const marginB=30;

ctx.strokeStyle = '#60a5fa'; ctx.lineWidth = 1.5; ctx.beginPath(); for (let i=0;i<data.length;i++){ const d = data[i]; const x = marginL + ((d.timestamp - t0)/(t1 - t0 || 1))*(w - marginL - marginR); const y = marginT + (1 - (d.value - minV)/(maxV - minV || 1))*(h - marginT - marginB); if (i===0) ctx.moveTo(x,y); else ctx.lineTo(x,y); } ctx.stroke(); } // HUD ctx.fillStyle = '#fff'; ctx.font = '12px Inter, system-ui'; ctx.fillText(Points: ${data.length}, 12, 14); ctx.fillText(FPS (approx): ${fps}, 12, 28); raf = requestAnimationFrame(render); } raf = requestAnimationFrame(render); return ()=>{ cancelAnimationFrame(raf); window.removeEventListener('resize', resize); }; },[data,fps]); return <div style={{width:'100%', height:360}}> <canvas ref={canvasRef} style={{width:'100%', height:'100%', borderRadius:8, display:'block'}} /> ; }

copy

--- components/ui/PerformanceMonitor.tsx

tsx 'use client'; import React from 'react'; import { useFPS } from '../../hooks/usePerformanceMonitor'; export default function PerformanceMonitor(){ const fps =

useFPS(); const mem = (performance as any).memory ? Math.round((performance as any).memory.usedJSHeapSize/1024/1024) + 'MB' : 'N/A'; return (<div style={{padding:12, background:'#fff', borderRadius:8}}> <h3>Performance</h3> FPS: {fps} Memory: {mem}); }

copy

--- components/controls/TimeRangeSelector.tsx

tsx 'use client'; import React from 'react'; export default function TimeRangeSelector(){ return (<div style={{padding:12, background:'#fff', borderRadius:8}}> <label>Time Range</label> <select style={{width:'100%', marginTop:6}} defaultValue={'60s'}> <option value={'15s'}>15 seconds</option> <option value={'60s'}>60 seconds</option> <option value={'5m'}>5 minutes</option> <option value={'1h'}>1 hour</option> </select>); }

copy

--- components/ui/DataTable.tsx

tsx 'use client'; import React from 'react'; import { useDataContext } from '../providers/DataProvider'; export default function DataTable(){ const { data } = useDataContext(); const rows = data.slice(-200).reverse(); return (<div style={{height:300, overflow:'auto', background:'#fff', borderRadius:8, padding:8}}> <table style={{width:'100%', borderCollapse:'collapse'}}> <thead><tr><th>Time</th><th>Value</th></tr></thead> <tbody> {rows.map((r,i)=> (<tr key={i}><td>{new Date(r.timestamp).toLocaleTimeString()}</td><td>{r.value.toFixed(2)}</td></tr>))} </tbody> </table>); }

copy

--- globals.css

css :root{ --bg:#0b1020; } html,body,#__next{ height:100%; } body{ background:linear-gradient(180deg,#071133 0%, #071326 100%); color:#e6eef8; margin:0; font-family:Inter,system-ui,Segoe UI,Roboto,Helvetica,Arial,sans-serif }

copy

--- package.json

json { "name": "performance-dashboard", "version": "1.0.0", "private": true, "scripts": { "dev": "next dev", "build": "next build", "start": "next start" }, "dependencies": { "next": "14.0.0", "react": "18.2.0", "react-dom": "18.2.0" }, "devDependencies": { "typescript": "5.4.2" } }

copy

--- tsconfig.json

json { "compilerOptions": { "target": "es2022", "lib": ["dom","dom.iterable","esnext"], "allowJs": true, "skipLibCheck": true, "strict": true, "forceConsistentCasingInFileNames": true, "module": "esnext", "moduleResolution": "node", "resolveJsonModule": true, "isolatedModules": true, "jsx": "react-jsx" } }

copy

--- README.md (short)

md # Performance Dashboard (Next.js + TypeScript) ## Quick start 1. npm install 2. npm run dev 3. Open http://localhost:3000/dashboard

This demo simulates a 100ms data stream and keeps a sliding window of points. The chart renders on a Canvas using requestAnimationFrame and aims to maintain 60fps with 10k+ points.

copy

--- PERFORMANCE.md (short)

md # PERFORMANCE NOTES - FPS measured via a small FPS hook; open Performance Monitor in UI. - Keep a sliding window, avoid re-allocating large arrays each tick. - Use canvas for drawing large point counts. - Further improvements: Web Worker for aggregation, OffscreenCanvas, level-of-detail rendering, binary buffers for data transfer.