JS | ATIVIDADE

window.addEventListener('error', e => console.error('ATIVIDADE HARD ERROR:', e.error || e.message));

import { db } from "../firebase-config.js";

import {

  collection, getDocs, query, orderBy, where,

  addDoc, serverTimestamp, deleteDoc, doc

} from "https://www.gstatic.com/firebasejs/10.12.0/firebase-firestore.js";

/\* ===============================

   Carregar Chart.js on-demand

   =============================== \*/

async function ensureChartJS(){

  if (window.Chart) return;

  await new Promise((resolve, reject) => {

    const s = document.createElement("script");

    s.src = "https://cdn.jsdelivr.net/npm/chart.js@4.4.3/dist/chart.umd.min.js";

    s.onload = resolve;

    s.onerror = reject;

    document.head.appendChild(s);

  });

}

/\* ===============================

   Paleta / Tema

   =============================== \*/

function isDark(){

  return document.documentElement.getAttribute("data-theme") === "dark";

}

function chartColors(){

  const dark = isDark();

  return {

    grid: dark ? "rgba(255,255,255,.12)" : "rgba(0,0,0,.12)",

    ticks: dark ? "rgba(255,255,255,.85)" : "rgba(0,0,0,.7)",

    title: dark ? "#fff" : "#111",

    tooltipBg: dark ? "rgba(17,17,17,.95)" : "rgba(255,255,255,.95)",

    tooltipFg: dark ? "#fff" : "#111",

  };

}

const PALETTE = [

  "#4F46E5","#22C55E","#EAB308","#EF4444","#06B6D4","#F59E0B",

  "#A855F7","#10B981","#3B82F6","#F472B6","#84CC16","#14B8A6"

];

/\* manter instâncias \*/

const charts = {

  setores:null, mercados:null, top5:null, top5Yield:null, timeline:null, divCal:null

};

/\* ===============================

   Helpers

   =============================== \*/

function toNumStrict(v){

  const n = Number(v);

  return Number.isFinite(n) ? n : NaN;

}

function isFiniteNum(v){

   if (v === null || v === undefined || v === "") return false;

   const n = Number(v);

   return Number.isFinite(n);

 }

function formatNum(n){ return Number(n || 0).toLocaleString("pt-PT"); }

/\*\* Carrega info complementar (valorStock, dividendo, periodicidade, mes, pe, sma, etc.) \*/

async function fetchDividendInfoByTickers(tickers){

  const out = new Map();

  const chunks = [];

  for (let i=0; i<tickers.length; i+=10) chunks.push(tickers.slice(i, i+10));

  for (const chunk of chunks){

    const q2 = query(collection(db, "acoesDividendos"), where("ticker", "in", chunk));

    const snap = await getDocs(q2);

    snap.forEach(docu => {

      const d = docu.data();

      if (d.ticker) out.set(String(d.ticker).toUpperCase(), d);

    });

  }

  return out;

}

function pickBestRate(info){

  if (typeof info?.taxaCrescimento\_1mes === "number") return { taxa: info.taxaCrescimento\_1mes, periodLabel: "mês" };

  if (typeof info?.taxaCrescimento\_1semana === "number") return { taxa: info.taxaCrescimento\_1semana, periodLabel: "semana" };

  if (typeof info?.taxaCrescimento\_1ano === "number") return { taxa: info.taxaCrescimento\_1ano, periodLabel: "ano" };

  return { taxa: null, periodLabel: null };

}

function estimateTime(currentPrice, targetPrice, growthPct, periodLabel) {

  const r = Number(growthPct || 0) / 100;

  if (r <= 0 || !isFiniteNum(currentPrice) || !isFiniteNum(targetPrice) || currentPrice <= 0 || targetPrice <= 0) return "—";

  const n = Math.log(targetPrice / currentPrice) / Math.log(1 + r);

  if (!isFinite(n) || n < 0) return "—";

  if (periodLabel === "semana") return `${n.toFixed(1)} semanas`;

  if (periodLabel === "mês") return `${n.toFixed(1)} meses`;

  return `${n.toFixed(1)} anos`;

}

/\* ---- dividendos & calendário ---- \*/

const MES\_IDX = {

  "janeiro":0,"fevereiro":1,"março":2,"marco":2,"abril":3,"maio":4,"junho":5,"julho":6,

  "agosto":7,"setembro":8,"outubro":9,"novembro":10,"dezembro":11

};

function pagamentosAno(periodicidade){

  const p = String(periodicidade || "").toLowerCase();

  if (p.startsWith("mensal")) return 12;

  if (p.startsWith("trimes")) return 4;

  if (p.startsWith("semes")) return 2;

  if (p.startsWith("anual"))  return 1;

  return 0;

}

function mesesPagos(periodicidade, mesTipico){

  const p = String(periodicidade || "").toLowerCase();

  const baseIdx = MES\_IDX[String(mesTipico||"").trim().toLowerCase()];

  if (p.startsWith("mensal")) return Array.from({length:12},(\_,i)=>i);

  if (p.startsWith("trimes")){

    const start = Number.isFinite(baseIdx) ? baseIdx : 0;

    return [start, (start+3)%12, (start+6)%12, (start+9)%12];

  }

  if (p.startsWith("semes")){

    const start = Number.isFinite(baseIdx) ? baseIdx : 0;

    return [start, (start+6)%12];

  }

  if (p.startsWith("anual")){

    return Number.isFinite(baseIdx) ? [baseIdx] : [];

  }

  return [];

}

/\* ===============================

   Renders — gráficos

   =============================== \*/

function renderSetorDoughnut(map){

  const el = document.getElementById("chartSetores");

  if (!el) return;

  charts.setores?.destroy();

  const labels = Array.from(map.keys());

  const data   = Array.from(map.values());

  if (!labels.length){ el.getContext("2d").clearRect(0,0,el.width, el.height); return; }

  charts.setores = new Chart(el, {

    type: "doughnut",

    data: { labels, datasets:[{ data, backgroundColor: labels.map((\_,i)=>PALETTE[i%PALETTE.length]), borderWidth:1 }] },

    options: {

      responsive:true, maintainAspectRatio:false, cutout:"62%",

      animation:{ animateRotate:false, animateScale:false }, // evita “tremer”

      plugins:{

        legend:{ position:"bottom", labels:{ color: chartColors().ticks } },

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.label}: €${Number(ctx.parsed).toLocaleString("pt-PT",{minimumFractionDigits:2})}` }

        }

      }

    }

  });

}

function renderMercadoDoughnut(map){

  const el = document.getElementById("chartMercados");

  if (!el) return;

  charts.mercados?.destroy();

  const labels = Array.from(map.keys());

  const data   = Array.from(map.values());

  if (!labels.length){ el.getContext("2d").clearRect(0,0,el.width, el.height); return; }

  charts.mercados = new Chart(el, {

    type: "doughnut",

    data: { labels, datasets:[{ data, backgroundColor: labels.map((\_,i)=>PALETTE[(i+5)%PALETTE.length]), borderWidth:1 }] },

    options: {

      responsive:true, maintainAspectRatio:false, cutout:"62%",

      animation:{ animateRotate:false, animateScale:false },

      plugins:{

        legend:{ position:"bottom", labels:{ color: chartColors().ticks } },

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.label}: €${Number(ctx.parsed).toLocaleString("pt-PT",{minimumFractionDigits:2})}` }

        }

      }

    }

  });

}

function renderTop5Bar(gruposArr){

  const el = document.getElementById("chartTop5");

  if (!el) return;

  charts.top5?.destroy();

  const ativos = gruposArr.filter(g=>g.qtd>0)

    .sort((a,b)=> (b.investido||0)-(a.investido||0)).slice(0,5);

  if (!ativos.length){ el.getContext("2d").clearRect(0,0,el.width, el.height); return; }

  const labels = ativos.map(a=>a.ticker);

  const invest = ativos.map(a=>a.investido||0);

  const lucro  = ativos.map(a=>a.lucroAtual||0);

  charts.top5 = new Chart(el, {

    type:"bar",

    data:{ labels, datasets:[

      { label:"Investido (€)", data:invest, backgroundColor:"#3B82F6" },

      { label:"Lucro Atual (€)", data:lucro, backgroundColor:"#22C55E" }

    ]},

    options:{

      responsive:true, maintainAspectRatio:false,

      scales:{ x:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid } },

               y:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid } } },

      plugins:{ legend:{ labels:{ color: chartColors().ticks }},

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.dataset.label}: €${Number(ctx.parsed.y||0).toLocaleString("pt-PT",{minimumFractionDigits:2})}` } }

      }

    }

  });

}

function renderTop5YieldBar(rows){

  const el = document.getElementById("chartTop5Yield");

  if (!el) return;

  charts.top5Yield?.destroy();

  const ativos = rows.filter(r=>r.active && isFiniteNum(r.yieldCur))

    .sort((a,b)=> b.yieldCur - a.yieldCur).slice(0,5);

  if (!ativos.length){ el.getContext("2d").clearRect(0,0,el.width, el.height); return; }

  const labels = ativos.map(a=>a.ticker);

  const ys     = ativos.map(a=>Number(a.yieldCur\*100).toFixed(2));

  charts.top5Yield = new Chart(el, {

    type:"bar",

    data:{ labels,

      datasets:[{ label:"Yield (%)", data:ys, backgroundColor:"#EAB308" }] },

    options:{

      indexAxis:"y", responsive:true, maintainAspectRatio:false,

      scales:{ x:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid } },

               y:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid } } },

      plugins:{ legend:{ labels:{ color: chartColors().ticks }},

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.dataset.label}: ${Number(ctx.parsed.x||0).toFixed(2)}%` } }

      }

    }

  });

}

function renderTimeline(points){

  const el = document.getElementById("chartTimeline");

  if (!el) return;

  charts.timeline?.destroy();

  if (!points.length){ el.getContext("2d").clearRect(0,0,el.width, el.height); return; }

  const labels   = points.map(p=>p.label);

  const invested = points.map(p=>p.cumInvest);

  const valueNow = points.map(p=>p.valueNow);

  charts.timeline = new Chart(el,{

    type:"line",

    data:{ labels, datasets:[

      { label:"Investido acumulado (€)", data:invested, tension:.25, borderWidth:2 },

      { label:"Avaliação atual (€)",     data:valueNow, tension:.25, borderWidth:2 }

    ]},

    options:{

      responsive:true, maintainAspectRatio:false, elements:{ point:{ radius:0 } },

      scales:{ x:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid }},

               y:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid }} },

      plugins:{ legend:{ labels:{ color: chartColors().ticks }},

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.dataset.label}: €${Number(ctx.parsed.y||0).toLocaleString("pt-PT",{minimumFractionDigits:2})}` } }

      }

    }

  });

}

function renderDividendoCalendario12m(arrEuros12){

  const el = document.getElementById("chartDivCalendario");

  if (!el) return;

  charts.divCal?.destroy();

  const labels = ["Jan","Fev","Mar","Abr","Mai","Jun","Jul","Ago","Set","Out","Nov","Dez"];

  charts.divCal = new Chart(el,{

    type:"bar",

    data:{ labels, datasets:[{ label:"€ / mês (estimado)", data:arrEuros12 }] },

    options:{

      responsive:true, maintainAspectRatio:false,

      scales:{ x:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid }},

               y:{ ticks:{ color: chartColors().ticks }, grid:{ color: chartColors().grid }} },

      plugins:{ legend:{ labels:{ color: chartColors().ticks }},

        tooltip:{ backgroundColor: chartColors().tooltipBg, titleColor: chartColors().tooltipFg, bodyColor: chartColors().tooltipFg,

          callbacks:{ label:(ctx)=>` ${ctx.dataset.label}: €${Number(ctx.parsed.y||0).toLocaleString("pt-PT",{minimumFractionDigits:2})}` } }

      }

    }

  });

}

/\* ===============================

   Quick Actions (comprar/vender)

   =============================== \*/

function wireQuickActions(gruposArr){

  const byTicker = new Map(gruposArr.map(g => [g.ticker, g]));

  const $ = s => document.querySelector(s);

  const modal   = $("#pfAddModal");

  const title   = $("#pfAddTitle");

  const form    = $("#pfAddForm");

  const close   = $("#pfAddClose");

  const cancel  = $("#pfAddCancel");

  const tipoSel = $("#pfTipoAcao");

  const labelP  = $("#pfLabelPreco");

  const vendaTotWrap = $("#pfVendaTotalWrap");

  const vendaTot = $("#pfVendaTotal");

  const fTicker = $("#pfTicker");

  const fNome   = $("#pfNome");

  const fSetor  = $("#pfSetor");

  const fMerc   = $("#pfMercado");

  const fQtd    = $("#pfQuantidade");

  const fPreco  = $("#pfPreco");

  const fObj    = $("#pfObjetivo");

  function open(kind, ticker){

    const g = byTicker.get(ticker);

    if (!g) return;

    modal?.classList.remove("hidden");

    title && (title.textContent = kind === "compra" ? "Comprar ativo" : "Vender ativo");

    tipoSel && (tipoSel.value = kind);

    if (fTicker) fTicker.value = g.ticker;

    if (fNome)   fNome.value   = g.nome;

    if (fSetor)  fSetor.value  = g.setor;

    if (fMerc)   fMerc.value   = g.mercado;

    if (fQtd)    fQtd.value    = "";

    if (fPreco)  fPreco.value  = "";

    if (fObj)    fObj.value    = g.objetivo || "";

    if (vendaTot) vendaTot.checked = false;

    if (vendaTotWrap) vendaTotWrap.style.display = kind === "venda" ? "block" : "none";

    if (labelP && labelP.firstChild) labelP.firstChild.textContent = kind === "venda" ? "Preço de venda (€)" : "Preço de compra (€)";

  }

  function closeModal(){

    modal?.classList.add("hidden");

    form?.reset();

    if (vendaTot) vendaTot.checked = false;

    if (vendaTotWrap) vendaTotWrap.style.display = "none";

    if (labelP && labelP.firstChild) labelP.firstChild.textContent = "Preço de compra (€)";

  }

  close?.addEventListener("click", closeModal);

  cancel?.addEventListener("click", closeModal);

  modal?.addEventListener("click", (e)=>{ if (e.target.id === "pfAddModal") closeModal(); });

  document.getElementById("listaAtividades")?.addEventListener("click", (e)=>{

    const buy = e.target.closest?.("[data-buy]");

    const sell = e.target.closest?.("[data-sell]");

    if (buy)  open("compra", buy.getAttribute("data-buy"));

    if (sell) open("venda",  sell.getAttribute("data-sell"));

  });

  tipoSel?.addEventListener("change", () => {

    const isVenda = tipoSel.value === "venda";

    if (labelP && labelP.firstChild) labelP.firstChild.textContent = isVenda ? "Preço de venda (€)" : "Preço de compra (€)";

    if (vendaTotWrap) vendaTotWrap.style.display = isVenda ? "block" : "none";

  });

  form?.addEventListener("submit", async (e) => {

    e.preventDefault();

    const tipo  = (tipoSel?.value || "compra").toLowerCase();

    const nome  = fNome?.value.trim() || "";

    const ticker= fTicker?.value.trim().toUpperCase() || "";

    const setor = fSetor?.value.trim() || "";

    const merc  = fMerc?.value.trim() || "";

    const qtd   = toNumStrict(fQtd?.value);

    const preco = toNumStrict(fPreco?.value);

    const obj   = toNumStrict(fObj?.value);

    const vendaTotal = !!vendaTot?.checked;

    if (!ticker || !nome || !Number.isFinite(qtd) || !Number.isFinite(preco) || qtd<=0 || preco<=0) {

      alert("Preenche Ticker, Nome, Quantidade (>0) e Preço (>0).");

      return;

    }

    const quantidade = tipo === "venda" ? -Math.abs(qtd) : Math.abs(qtd);

    const payload = {

      tipoAcao: tipo, nome, ticker, setor, mercado: merc,

      quantidade, precoCompra: preco,

      objetivoFinanceiro: Number.isFinite(obj) ? obj : 0,

      dataCompra: serverTimestamp(),

    };

    try {

      await addDoc(collection(db, "ativos"), payload);

      if (tipo === "venda" && vendaTotal) {

        const toDelQ = query(collection(db, "ativos"), where("ticker", "==", ticker));

        const snapDel = await getDocs(toDelQ);

        const promises = [];

        snapDel.forEach(d => promises.push(deleteDoc(doc(db, "ativos", d.id))));

        await Promise.all(promises);

      }

      closeModal();

      location.reload();

    } catch (err) {

      console.error("❌ Erro ao guardar movimento:", err);

      alert("Não foi possível guardar. Tenta novamente.");

    }

  });

}

/\* ===============================

   INIT (screen)

   =============================== \*/

export async function initScreen() {

  const cont = document.getElementById("listaAtividades");

  if (!cont) return;

  cont.innerHTML = "A carregar…";

  await ensureChartJS();

  try {

    // 1) Buscar movimentos e agrupar por ticker

    const qSnap = await getDocs(

      query(collection(db, "ativos"), orderBy("dataCompra", "desc"))

    );

    if (qSnap.empty) {

      cont.innerHTML = `<p class="muted">Sem atividades ainda.</p>`;

      renderSetorDoughnut(new Map());

      renderMercadoDoughnut(new Map());

      renderTop5Bar([]);

      renderTop5YieldBar([]);

      renderTimeline([]);

      renderDividendoCalendario12m(new Array(12).fill(0));

      return;

    }

    const grupos = new Map();

    const movimentosAsc = [];

    qSnap.forEach((docu) => {

      const d = docu.data();

      const ticker = String(d.ticker || "").toUpperCase();

      if (!ticker) return;

      const qtd   = toNumStrict(d.quantidade);

      const preco = toNumStrict(d.precoCompra);

      const safeQtd   = Number.isFinite(qtd)   ? qtd   : 0;

      const safePreco = Number.isFinite(preco) ? preco : 0;

      const invest = safeQtd \* safePreco;

      const g = grupos.get(ticker) || {

        ticker,

        nome: d.nome || ticker,

        setor: d.setor || "-",

        mercado: d.mercado || "-",

        qtd: 0,

        investido: 0,

        objetivo: 0,

        anyObjSet: false,

        lastDate: null,

      };

      g.qtd       += safeQtd;

      g.investido += Number.isFinite(invest) ? invest : 0;

      const obj = toNumStrict(d.objetivoFinanceiro);

      if (!g.anyObjSet && Number.isFinite(obj) && obj > 0) {

        g.objetivo = obj;

        g.anyObjSet = true;

      }

      const dt =

        d.dataCompra && typeof d.dataCompra.toDate === "function"

          ? d.dataCompra.toDate()

          : null;

      if (!g.lastDate || (dt && dt > g.lastDate)) g.lastDate = dt;

      g.nome = d.nome || g.nome;

      g.setor = d.setor || g.setor;

      g.mercado = d.mercado || g.mercado;

      grupos.set(ticker, g);

      movimentosAsc.push({ date: dt || new Date(0), ticker, qtd: safeQtd, preco: safePreco });

    });

    const gruposArr = Array.from(grupos.values());

    // Debug opcional

    const dbgEl = document.getElementById("prtDebug");

    if (dbgEl) {

      const totalMovs = qSnap.size;

      const tickers = gruposArr.length;

      const abertos = gruposArr.filter(g => g.qtd > 0).length;

      dbgEl.textContent = `Movimentos: ${totalMovs} | Tickers: ${tickers} | Abertos: ${abertos}`;

    }

    console.log("[atividade] grupos:", gruposArr);

    // 2) Info de cota / dividendos

    const tickers = gruposArr.map((g) => g.ticker);

    const infoMap = await fetchDividendInfoByTickers(tickers);

    const fmtEUR = new Intl.NumberFormat("pt-PT", { style: "currency", currency: "EUR" });

    const fmtDate = new Intl.DateTimeFormat("pt-PT", { year: "numeric", month: "short", day: "2-digit" });

    // Enriquecer grupos com métricas para gráficos/cards

    const rowsForYield = []; // para Top5 Yield

    gruposArr.forEach((g) => {

      const info = infoMap.get(g.ticker) || {};

      const precoAtual = isFiniteNum(info.valorStock) ? Number(info.valorStock) : null;

      const precoMedio = g.qtd !== 0 ? (g.investido / (g.qtd || 1)) : 0; // evita div/0

      g.lucroAtual = precoAtual !== null ? (precoAtual - precoMedio) \* g.qtd : 0;

      g.precoAtual = precoAtual;

      const dividendo = isFiniteNum(info.dividendo) ? Number(info.dividendo) : 0; // por pagamento (unit.)

      const dmed24 = isFiniteNum(info.dividendoMedio24m) ? Number(info.dividendoMedio24m) : 0; // anual (média 24m)

      const pe = isFiniteNum(info.peRatio) ? Number(info.peRatio) : null;

      const sma50 = isFiniteNum(info.sma50) ? Number(info.sma50) : null;

      const sma200 = isFiniteNum(info.sma200) ? Number(info.sma200) : null;

      const yCur = precoAtual && dividendo > 0 && info.periodicidade

        ? (dividendo \* pagamentosAno(info.periodicidade)) / precoAtual

        : (precoAtual && dmed24 > 0 ? dmed24 / precoAtual : null);

      const y24m = precoAtual && dmed24 > 0 ? dmed24 / precoAtual : null;

      g.\_yCur = yCur;

      g.\_y24m = y24m;

      g.\_pe = pe;

      g.\_sma50 = sma50;

      g.\_sma200 = sma200;

      rowsForYield.push({ ticker: g.ticker, active: g.qtd > 0, yieldCur: yCur });

    });

    // 2.1) Distribuições (apenas posições abertas)

    const setoresMap = new Map();

    const mercadosMap = new Map();

    for (const g of gruposArr) {

      if ((g.qtd || 0) <= 0) continue;

      const setor = g.setor || "—";

      setoresMap.set(setor, (setoresMap.get(setor) || 0) + (g.investido || 0));

      const merc = g.mercado || "—";

      mercadosMap.set(merc, (mercadosMap.get(merc) || 0) + (g.investido || 0));

    }

    // 2.2) KPIs agregados (apenas posições abertas)

    const abertos = gruposArr.filter(g => g.qtd > 0);

    const totalInvestido = abertos.reduce((a, g) => a + (g.investido || 0), 0);

    const lucroTotal     = abertos.reduce((a, g) => a + (g.lucroAtual || 0), 0);

    const retornoPct     = totalInvestido ? (lucroTotal / totalInvestido) \* 100 : 0;

    // rendimento anual esperado (unitário \* pagamentos \* qtd)

    let rendimentoAnual = 0;

    const eurosMes = new Array(12).fill(0); // calendário 12m

    for (const g of abertos) {

      const info = infoMap.get(g.ticker) || {};

      const divUnit = isFiniteNum(info.dividendo) ? Number(info.dividendo) : 0; // por pagamento

      const per = info.periodicidade;

      const mesT = info.mes;

      const payN = pagamentosAno(per);

      const anualTicker = g.qtd \* divUnit \* payN;

      rendimentoAnual += anualTicker;

      // distribuir pelos meses estimados (cada mês recebe qtd \* unidade)

      const meses = mesesPagos(per, mesT);

      for (const m of meses) eurosMes[m] += g.qtd \* divUnit;

    }

    // exposição acima da SMA200 (peso por investido)

    let somaPesosAcima = 0;

    for (const g of abertos) {

      if (!totalInvestido) continue;

      const w = (g.investido || 0) / totalInvestido;

      const p = g.precoAtual, s200 = g.\_sma200;

      if (isFiniteNum(p) && isFiniteNum(s200) && Number(p) > Number(s200)) somaPesosAcima += w;

    }

    const expSMA200Pct = somaPesosAcima \* 100;

    // preencher KPIs

    const elTI = document.getElementById("prtTotalInvestido");

    const elRA = document.getElementById("prtRendimentoAnual");

    const elRP = document.getElementById("prtRetorno");

    const elEX = document.getElementById("prtExpSMA200");

    if (elTI) elTI.textContent = fmtEUR.format(totalInvestido);

    if (elRA) elRA.textContent = fmtEUR.format(rendimentoAnual);

    if (elRP) elRP.textContent = `${retornoPct.toFixed(1)}%`;

    if (elEX) elEX.textContent = `${expSMA200Pct.toFixed(0)}%`;

    // 2.3) Timeline (compras/vendas acumuladas; avaliação com preços atuais)

    movimentosAsc.sort((a, b) => a.date - b.date);

    const qtyNow = new Map();

    const priceNow = new Map();

    gruposArr.forEach((g) => {

      if (isFiniteNum(g.precoAtual)) priceNow.set(g.ticker, Number(g.precoAtual));

      qtyNow.set(g.ticker, 0);

    });

    let cumInvest = 0;

    const timelinePoints = [];

    for (const m of movimentosAsc) {

      const deltaInvest = Number(m.qtd) \* Number(m.preco);

      cumInvest += deltaInvest;

      qtyNow.set(m.ticker, (qtyNow.get(m.ticker) || 0) + m.qtd);

      let valueNow = 0;

      qtyNow.forEach((q, tk) => {

        const p = priceNow.get(tk);

        if (isFiniteNum(p)) valueNow += q \* Number(p);

      });

      timelinePoints.push({

        label: isFinite(m.date?.getTime?.()) ? fmtDate.format(m.date) : "",

        cumInvest: cumInvest,

        valueNow: valueNow,

      });

    }

    // 3) Render GRÁFICOS

    renderSetorDoughnut(setoresMap);

    renderMercadoDoughnut(mercadosMap);

    renderTop5Bar(gruposArr);

    renderTop5YieldBar(rowsForYield);

    renderTimeline(timelinePoints);

    renderDividendoCalendario12m(eurosMes);

    // 4) Render LISTA — mostra todos os tickers, com estado

    const html = gruposArr

      .filter((g) => Number.isFinite(g.qtd))

      .map((g) => {

        const info = infoMap.get(g.ticker) || {};

        const precoAtual = g.precoAtual;

        const precoMedio = g.qtd !== 0 ? (g.investido / (g.qtd || 1)) : 0;

        const lucroAtual = g.lucroAtual || 0;

        // estado posição

        const posStatus = g.qtd > 0 ? "Posição aberta" : (g.qtd < 0 ? "Posição negativa (ver movimentos)" : "Posição encerrada");

        const itemClass = g.qtd > 0 ? "" : " muted";

        // progresso objetivo (barra zero ao centro)

        let pctText = "—";

        let barHTML = "";

        const objetivo = g.objetivo > 0 ? g.objetivo : 0;

        if (objetivo > 0) {

          const progresso = (lucroAtual / objetivo) \* 100;

          const clamped = Math.max(-100, Math.min(100, progresso));

          const sideWidthPct = (Math.abs(clamped) / 2).toFixed(1);

          const positive = clamped >= 0;

          pctText = `${clamped.toFixed(0)}%`;

          barHTML = `

            <div class="progress-dual">

              <div class="track">

                <div class="fill ${positive ? "positive" : "negative"}" style="width:${sideWidthPct}%"></div>

                <div class="zero"></div>

              </div>

            </div>

          `;

        }

        const tp2Necessario =

          objetivo > 0 && g.qtd !== 0 ? precoMedio + objetivo / (g.qtd || 1) : null;

        const { taxa, periodLabel } = pickBestRate(info);

        const estimativa =

          tp2Necessario && precoAtual

            ? estimateTime(precoAtual, tp2Necessario, taxa, periodLabel)

            : "—";

        const dataTxt = g.lastDate ? fmtDate.format(g.lastDate) : "sem data";

        // badges

        const dividendoUnit = isFiniteNum(info.dividendo) ? Number(info.dividendo) : 0;

        const payN = pagamentosAno(info.periodicidade);

        const divAnualEst = dividendoUnit \* payN;

        const yCur = g.\_yCur; // 0..1

        const y24 = g.\_y24m; // 0..1

        const pe = g.\_pe;

        const s50 = g.\_sma50;

        const s200 = g.\_sma200;

        const yPct = isFiniteNum(yCur) ? (yCur\*100).toFixed(2) + "%" : "—";

        const y24Pct = isFiniteNum(y24) ? (y24\*100).toFixed(2) + "%" : "—";

        const yBadge =

          isFiniteNum(yCur) && isFiniteNum(y24)

            ? (yCur > y24 ? "↑ acima da média" : "↓ abaixo da média")

            : "";

        const peBadge = isFiniteNum(pe)

          ? (pe < 15 ? "Barato" : (pe <= 25 ? "Justo" : "Caro"))

          : "—";

        const d50 =

          isFiniteNum(s50) && isFiniteNum(precoAtual)

            ? ((precoAtual - s50) / s50) \* 100

            : null;

        const d200 =

          isFiniteNum(s200) && isFiniteNum(precoAtual)

            ? ((precoAtual - s200) / s200) \* 100

            : null;

        const d50Txt = d50 === null ? "—" : `${d50.toFixed(1)}%`;

        const d200Txt = d200 === null ? "—" : `${d200.toFixed(1)}%`;

        const stopLight =

          isFiniteNum(precoAtual) && isFiniteNum(s200) && precoAtual < s200 && Number(info.taxaCrescimento\_1mes || 0) < 0

            ? "🔴"

            : ((isFiniteNum(precoAtual) && isFiniteNum(s200) && precoAtual < s200) || Number(info.taxaCrescimento\_1mes || 0) < 0)

            ? "🟡"

            : (isFiniteNum(precoAtual) && isFiniteNum(s200) && precoAtual > s200 && Number(info.taxaCrescimento\_1mes || 0) > 0)

            ? "🟢"

            : "⚪️";

        // ações — esconder vender em posição encerrada/negativa

        const actions = g.qtd > 0

          ? `<div class="actions-row" style="margin-top:.5rem">

               <button class="btn outline" data-buy="${g.ticker}">Comprar</button>

               <button class="btn ghost"  data-sell="${g.ticker}">Vender</button>

             </div>`

          : `<div class="actions-row" style="margin-top:.5rem">

               <button class="btn outline" data-buy="${g.ticker}">Reabrir (comprar)</button>

             </div>`;

        // linha de análise extra

        const analysis = `

          <p class="muted" style="margin-top:.4rem">

            ${stopLight} Yield: <strong>${yPct}</strong> (${yBadge || "—"}) •

            Yield 24m: <strong>${y24Pct}</strong> •

            P/E: <strong>${

              isFiniteNum(pe) ? Number(pe).toFixed(2) : "—"

            } (${peBadge})</strong> •

            Δ50d: <strong>${d50Txt}</strong> •

            Δ200d: <strong>${d200Txt}</strong>

          </p>

          <p class="muted">

            ${String(info.periodicidade || "n/A")} • paga em <strong>${String(

          info.mes || "n/A"

        )}</strong> •

            Div. unit.: <strong>${

              isFiniteNum(dividendoUnit) ? fmtEUR.format(dividendoUnit) : "—"

            }</strong> •

            Div. anual est.: <strong>${

              isFiniteNum(divAnualEst) ? fmtEUR.format(divAnualEst) : "—"

            }</strong>

          </p>

        `;

        return `

          <div class="activity-item${itemClass}">

            <div class="activity-left">

              <span class="activity-icon">${g.qtd > 0 ? "📦" : "📭"}</span>

              <div>

                <p><strong>${g.nome}</strong> <span class="muted">(${g.ticker})</span></p>

                <p class="muted">${g.setor} • ${g.mercado} • ${posStatus}</p>

                <p class="muted">Última compra: ${dataTxt}</p>

                <p class="muted">

                  Qtd: <strong>${formatNum(g.qtd)}</strong> ·

                  Preço médio: <strong>${fmtEUR.format(precoMedio || 0)}</strong> ·

                  Preço atual: <strong>${precoAtual !== null ? fmtEUR.format(precoAtual) : "—"}</strong>

                </p>

                <p class="muted">

                  Investido: <strong>${fmtEUR.format(g.investido || 0)}</strong> ·

                  Lucro atual: <strong>${fmtEUR.format(lucroAtual)}</strong>

                </p>

                ${objetivo > 0 ? `

                  <div class="activity-meta">

                    <span>Objetivo (lucro): <strong>${fmtEUR.format(objetivo)}</strong></span>

                    <span>${pctText}</span>

                  </div>

                  ${barHTML}

                  <p class="muted">

                    TP2 necessário: <strong>${tp2Necessario ? fmtEUR.format(tp2Necessario) : "—"}</strong>

                    ${taxa !== null ? `· Estimativa: <strong>${estimativa}</strong>` : ``}

                  </p>

                ` : `<p class="muted">Sem objetivo definido para este ticker.</p>`}

                ${analysis}

                ${actions}

              </div>

            </div>

          </div>

        `;

      });

    cont.innerHTML = html.join("");

    // 5) Quick actions

    wireQuickActions(gruposArr);

    // 6) Re-render ao mudar de tema + limpeza

    if (window.\_\_prtThemeHandler) {

      window.removeEventListener("app:theme-changed", window.\_\_prtThemeHandler);

    }

    window.\_\_prtThemeHandler = () => {

      if (!document.getElementById("chartSetores")) return;

      renderSetorDoughnut(setoresMap);

      renderMercadoDoughnut(mercadosMap);

      renderTop5Bar(gruposArr);

      renderTop5YieldBar(rowsForYield);

      renderTimeline(timelinePoints);

      renderDividendoCalendario12m(eurosMes);

    };

    window.addEventListener("app:theme-changed", window.\_\_prtThemeHandler);

    // 7) Auto-clean quando saíres do ecrã

    const observer = new MutationObserver(() => {

      if (!document.getElementById("chartSetores")) {

        charts.setores?.destroy?.();

        charts.mercados?.destroy?.();

        charts.top5?.destroy?.();

        charts.top5Yield?.destroy?.();

        charts.timeline?.destroy?.();

        charts.divCal?.destroy?.();

        if (window.\_\_prtThemeHandler) {

          window.removeEventListener("app:theme-changed", window.\_\_prtThemeHandler);

          window.\_\_prtThemeHandler = null;

        }

        observer.disconnect();

      }

    });

    observer.observe(document.body, { childList: true, subtree: true });

  } catch (e) {

    console.error("Erro ao carregar atividades:", e);

    cont.innerHTML = `<p class="muted">Não foi possível carregar a lista.</p>`;

    renderSetorDoughnut(new Map());

    renderMercadoDoughnut(new Map());

    renderTop5Bar([]);

    renderTop5YieldBar([]);

    renderTimeline([]);

    renderDividendoCalendario12m(new Array(12).fill(0));

  }

}

/\* ========= Ajuda das Métricas (popup) ========= \*/

const HELP\_KEY = "prt.help.dismissed";

function wirePortfolioHelpModal() {

  const modal   = document.getElementById("prtHelpModal");

  if (!modal || modal.\_\_wired) return;

  modal.\_\_wired = true;

  const closeBtn = document.getElementById("prtHelpClose");

  const okBtn    = document.getElementById("prtHelpOK");

  const laterBtn = document.getElementById("prtHelpLater");

  const dontShow = document.getElementById("prtHelpDontShow");

  const close = (persist) => {

    if (persist && dontShow?.checked) {

      try { localStorage.setItem(HELP\_KEY, "1"); } catch {}

    }

    modal.classList.add("hidden");

  };

  closeBtn?.addEventListener("click", () => close(false));

  laterBtn?.addEventListener("click", () => close(false));

  okBtn?.addEventListener("click", () => close(true));

  // fecha ao clicar fora do diálogo

  modal.addEventListener("click", (e) => {

    if (e.target === modal) close(false);

  });

  // fecha com ESC

  document.addEventListener("keydown", (e) => {

    if (modal.classList.contains("hidden")) return;

    if (e.key === "Escape") close(false);

  });

}

function showPortfolioHelp(force = false) {

  const modal = document.getElementById("prtHelpModal");

  if (!modal) return;

  if (!force) {

    try { if (localStorage.getItem(HELP\_KEY) === "1") return; } catch {}

  }

  modal.classList.remove("hidden");

}