JS | DASHBOARD

// js/screens/dashboard.js

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

import {

  getDocs, collection, query, orderBy, limit, addDoc, serverTimestamp

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

export async function initScreen() {

  console.log("✅ dashboard.js iniciado");

  // --- ELEMENTOS DA UI ---

  const valorTotalEl    = document.getElementById("valorTotal");

  const retornoEl       = document.getElementById("retornoTotal");

  const lucroTotalEl    = document.getElementById("lucroTotal");

  const posicoesEl      = document.getElementById("posicoesAtivas");

  const objetivosEl     = document.getElementById("objetivosFinanceiros");

  const taxaSucessoEl   = document.getElementById("taxaSucesso");

  const objetivoTotalEl = document.getElementById("objetivoTotal");

  const valorCarteiraEl = document.getElementById("valorCarteira");

  // --- ACUMULADORES ---

  let totalInvestido = 0;

  let totalLucro = 0;

  let objetivoFinanceiroTotal = 0;

  let objetivosAtingidos = 0;

  try {

    // 1) Buscar ativos e preços atuais

    const [ativosSnapshot, acoesSnapshot] = await Promise.all([

      getDocs(collection(db, "ativos")),

      getDocs(collection(db, "acoesDividendos")),

    ]);

    // Mapa de preços atuais por TICKER

    const valorAtualMap = new Map();

    acoesSnapshot.forEach((doc) => {

      const d = doc.data();

      if (d.ticker && typeof d.valorStock === "number") {

        valorAtualMap.set(String(d.ticker).toUpperCase(), d.valorStock);

      }

    });

    // Agrupar ativos por TICKER

    const agrupadoPorTicker = new Map();

    ativosSnapshot.forEach((doc) => {

      const a = doc.data();

      const ticker = (a.ticker || "").toUpperCase();

      if (!ticker) return;

      const grupo = agrupadoPorTicker.get(ticker) || {

        quantidade: 0,

        investimento: 0,

        objetivoFinanceiro: 0,

        objetivoDefinido: false,

      };

      const quantidade   = parseFloat(a.quantidade || 0);

      const precoCompra  = parseFloat(a.precoCompra || 0);

      const objetivo     = parseFloat(a.objetivoFinanceiro || 0);

      grupo.quantidade   += quantidade;

      grupo.investimento += precoCompra \* quantidade;

      // objetivo conta uma única vez por ticker

      if (!grupo.objetivoDefinido && objetivo > 0) {

        grupo.objetivoFinanceiro = objetivo;

        grupo.objetivoDefinido   = true;

      }

      agrupadoPorTicker.set(ticker, grupo);

    });

    let totalObjetivos = 0;

    // Calcular KPIs globais

    agrupadoPorTicker.forEach((g, ticker) => {

      const precoAtual = valorAtualMap.get(ticker) || 0;

      const atual      = g.quantidade \* precoAtual;

      const lucro      = atual - g.investimento;

      totalInvestido += g.investimento;

      totalLucro     += lucro;

      if (g.objetivoDefinido) {

        totalObjetivos++;

        objetivoFinanceiroTotal += g.objetivoFinanceiro;

        if (lucro >= g.objetivoFinanceiro) objetivosAtingidos++;

      }

    });

    const retorno     = totalInvestido > 0 ? (totalLucro / totalInvestido) \* 100 : 0;

    const taxaSucesso = objetivoFinanceiroTotal > 0 ? (totalLucro / objetivoFinanceiroTotal) \* 100 : 0;

    const valorCarteira = totalInvestido + totalLucro; // 💰 investido + lucro/prejuízo

    // Atualizar UI

    if (valorTotalEl)     valorTotalEl.textContent     = `€${totalInvestido.toFixed(2)}`;

    if (lucroTotalEl)     lucroTotalEl.textContent     = `€${totalLucro.toFixed(2)}`;

    if (retornoEl)        retornoEl.textContent        = `${retorno.toFixed(1)}%`;

    if (posicoesEl)       posicoesEl.textContent       = agrupadoPorTicker.size;

    if (objetivosEl)      objetivosEl.textContent      = `${objetivosAtingidos}/${totalObjetivos}`;

    if (objetivoTotalEl)  objetivoTotalEl.textContent  = `€${objetivoFinanceiroTotal.toFixed(2)}`;

    if (taxaSucessoEl)    taxaSucessoEl.textContent    = `${taxaSucesso.toFixed(1)}%`;

    if (valorCarteiraEl)  valorCarteiraEl.textContent  = `€${valorCarteira.toFixed(2)} valor em carteira`;

  } catch (err) {

    console.error("❌ Erro nos KPIs:", err);

  }

  // 2) Atividades recentes (apenas campos solicitados, sem barras) — com expand/colapse

  await carregarAtividadeRecenteSimplificada();

  // 3) Botões

  document.getElementById("btnOportunidades")?.addEventListener("click", openOportunidades);

  document.getElementById("opClose")?.addEventListener("click", closeOportunidades);

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

    if (e.target.id === "opModal") closeOportunidades();

  });

  // Chips do período no popup

  document.querySelectorAll("#opModal .chip").forEach(btn => {

    btn.addEventListener("click", () => {

      const p = btn.getAttribute("data-periodo");

      opPeriodoAtual = p || "1s";

      setActiveChip(opPeriodoAtual);

      carregarTop10Crescimento(opPeriodoAtual);

    });

  });

    // --- Modal: Adicionar Ação ---

  const btnAdd = document.getElementById("btnAddAcao");

  const addModal = document.getElementById("addModal");

  const addClose = document.getElementById("addClose");

  const addCancel = document.getElementById("addCancel");

  const addForm = document.getElementById("addForm");

  const tipoAcaoSel = document.getElementById("tipoAcao");

  const labelPreco = document.getElementById("labelPreco");

  // abre

  btnAdd?.addEventListener("click", () => {

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

  });

  // fecha

  function closeAddModal() {

    addModal?.classList.add("hidden");

    addForm?.reset();

    // repõe o label (caso tenha mudado para Venda)

    if (labelPreco) labelPreco.firstChild.textContent = "Preço da transação (€)";

  }

  addClose?.addEventListener("click", closeAddModal);

  addCancel?.addEventListener("click", closeAddModal);

  addModal?.addEventListener("click", (e) => {

    if (e.target.id === "addModal") closeAddModal();

  });

  // muda o label conforme compra/venda (puramente visual)

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

    if (!labelPreco) return;

    if (tipoAcaoSel.value === "venda") {

      labelPreco.firstChild.textContent = "Preço de venda (€)";

    } else {

      labelPreco.firstChild.textContent = "Preço de compra (€)";

    }

  });

  // submit

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

    e.preventDefault();

    const tipoAcao = (document.getElementById("tipoAcao")?.value || "compra").toLowerCase();

    const nome     = (document.getElementById("nomeAtivo")?.value || "").trim();

    const ticker   = (document.getElementById("tickerAtivo")?.value || "").trim().toUpperCase();

    const setor    = (document.getElementById("setorAtivo")?.value || "").trim();

    const mercado  = (document.getElementById("mercadoAtivo")?.value || "").trim();

    const qtdRaw   = Number(document.getElementById("quantidadeAtivo")?.value || 0);

    const preco    = Number(document.getElementById("precoAtivo")?.value || 0);

    const objetivo = Number(document.getElementById("objetivoAtivo")?.value || 0);

    if (!nome || !ticker || !qtdRaw || !preco) {

      alert("Preenche pelo menos: Tipo, Nome, Ticker, Quantidade e Preço.");

      return;

    }

    // ── regra simples: vendas entram como quantidade negativa e usam o mesmo campo 'precoCompra'

    const quantidade = tipoAcao === "venda" ? -Math.abs(qtdRaw) : Math.abs(qtdRaw);

    const payload = {

      tipoAcao,              // "compra" ou "venda" (útil para auditoria)

      nome,

      ticker,

      setor,

      mercado,

      quantidade,            // negativo na venda

      precoCompra: preco,    // mantém a compatibilidade com os teus cálculos atuais

      objetivoFinanceiro: isNaN(objetivo) ? 0 : objetivo,

      dataCompra: serverTimestamp(), // data/hora automática

    };

    try {

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

      closeAddModal();

      // Atualiza o painel rapidamente. Para máxima simplicidade, recarrega:

      // (se preferires sem refresh, dá para chamar as funções de KPI e atividade novamente)

      window.location.reload();

    } catch (err) {

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

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

    }

  });

  wireDashBuy();

}

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

   ATIVIDADE RECENTE (SIMPLIFICADA) + EXPAND/COLAPSE

   ========================= \*/

let atividadesCache = [];     // guarda todos os docs formatados

let atividadesExpandido = false; // estado de expansão

async function carregarAtividadeRecenteSimplificada() {

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

  if (!cont) return;

  try {

    // Trazemos MAIS do que 4 para já termos tudo em cache

    const q = query(collection(db, "ativos"), orderBy("dataCompra", "desc"), limit(50));

    const snapAtivos = await getDocs(q);

    if (snapAtivos.empty) {

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

      return;

    }

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

    const fmtDateL = new Intl.DateTimeFormat("pt-PT", {

      year: "numeric", month: "long", day: "numeric",

      hour: "2-digit", minute: "2-digit", second: "2-digit",

      timeZoneName: "short"

    });

    atividadesCache = []; // reset cache

    snapAtivos.forEach(doc => {

        const d = doc.data();

        const nome        = d.nome || d.ticker || "Ativo";

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

        const setor       = d.setor || "-";

        const mercado     = d.mercado || "-";

        const precoCompra = Number(d.precoCompra || 0);

        const quantidade  = Number(d.quantidade || 0);

        // NOVO → badge COMPRA/VENDA

        const tipo  = (d.tipoAcao || "compra").toLowerCase();

        const badge = tipo === "venda"

          ? '<span class="tag venda">VENDA</span>'

          : '<span class="tag compra">COMPRA</span>';

        let dataTxt = "sem data";

        if (d.dataCompra && typeof d.dataCompra.toDate === "function") {

          dataTxt = fmtDateL.format(d.dataCompra.toDate());

        }

      atividadesCache.push(`

        <div class="activity-item">

          <div class="activity-left">

            <span class="activity-icon">🛒</span>

            <div>

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

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

              <p class="muted">${quantidade} ${quantidade === 1 ? "ação" : "ações"} @ ${fmtEUR.format(precoCompra)}</p>

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

            </div>

          </div>

        </div>

      `);

    });

    // Render inicial: só 2

    renderAtividades(cont, 2);

    atividadesExpandido = false;

    // Ligar o botão "Ver Toda Atividade" (sem mexer no HTML: apanha o primeiro .btn.outline.full nessa card)

    const btnVerTodos = document.querySelector(".dashboard .card.glass .btn.outline.full");

    if (btnVerTodos) {

      btnVerTodos.textContent = "Ver Toda Atividade";

      btnVerTodos.onclick = () => {

        atividadesExpandido = !atividadesExpandido;

        if (atividadesExpandido) {

          renderAtividades(cont, atividadesCache.length);

          btnVerTodos.textContent = "Mostrar menos";

        } else {

          renderAtividades(cont, 2);

          btnVerTodos.textContent = "Ver Toda Atividade";

        }

      };

    }

  } catch (e) {

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

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

  }

}

function renderAtividades(container, howMany) {

  const slice = atividadesCache.slice(0, howMany);

  container.innerHTML = slice.join("");

}

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

   POPUP: TOP 10 OPORTUNIDADES

   ========================= \*/

let opInterval = null;

let opPeriodoAtual = "1s";

async function carregarTop10Crescimento(periodo = "1s") {

  const lista = document.getElementById("listaTop10");

  if (!lista) return;

  lista.innerHTML = "🔄 A carregar...";

  const campos = {

    "1s":  "taxaCrescimento\_1semana",

    "1m":  "taxaCrescimento\_1mes",

    "1ano":"taxaCrescimento\_1ano",

  };

  const campo = campos[periodo] || campos["1s"];

  try {

    const snap = await getDocs(collection(db, "acoesDividendos"));

    const arr = [];

    snap.forEach(doc => {

      const d = doc.data();

      const crescimento = Number(d[campo] ?? 0);

      if (Number.isFinite(crescimento) && d.ticker) {

        arr.push({

          nome: d.nome || d.ticker,

          ticker: String(d.ticker).toUpperCase(),

          crescimento

        });

      }

    });

    const top10 = arr.sort((a,b) => b.crescimento - a.crescimento).slice(0, 10);

    if (top10.length === 0) {

      lista.innerHTML = "<li>😕 Nenhuma ação com crescimento positivo.</li>";

      return;

    }

    lista.innerHTML = top10.map(item => `

      <li>

        <div class="left">

          <strong>${item.nome}</strong>

          <span class="ticker">(${item.ticker})</span>

        </div>

        <span class="${item.crescimento >= 0 ? "gain" : "loss"}">

          ${item.crescimento >= 0 ? "+" : ""}${item.crescimento.toFixed(2)}%

        </span>

      </li>

    `).join("");

  } catch (err) {

    console.error("Erro ao carregar Top 10:", err);

    lista.innerHTML = "<li style='color:#f88;'>Erro ao carregar dados.</li>";

  }

}

function openOportunidades() {

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

  if (!modal) return;

  modal.classList.remove("hidden");

  setActiveChip(opPeriodoAtual);

  carregarTop10Crescimento(opPeriodoAtual);

  clearInterval(opInterval);

  opInterval = setInterval(() => {

    carregarTop10Crescimento(opPeriodoAtual);

  }, 30000);

}

function closeOportunidades() {

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

  if (!modal) return;

  modal.classList.add("hidden");

  clearInterval(opInterval);

  opInterval = null;

}

function setActiveChip(periodo) {

  document.querySelectorAll("#opModal .chip").forEach(ch => {

    const p = ch.getAttribute("data-periodo");

    ch.classList.toggle("active", p === periodo);

  });

}

// =========================

// Simulador Botão Definir Objetivo (versão wizard + dropdowns)

// =========================

// ---------- Helpers comuns ----------

function keyStr(s){

  return String(s ?? "")

    .normalize("NFD")

    .replace(/\p{Diacritic}/gu, "") // requer JS moderno

    .trim()

    .toLowerCase();

}

function euro(v){ return new Intl.NumberFormat("pt-PT",{style:"currency",currency:"EUR"}).format(v||0); }

function clamp(v,min,max){ return Math.max(min, Math.min(max, v)); }

function dividirPeriodicidade(dividendo, periodicidade){

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

  if (p === "mensal")     return dividendo \* 12;

  if (p === "trimestral") return dividendo \* 4;

  if (p === "semestral")  return dividendo \* 2;

  return dividendo; // anual ou n/a

}

function campoCrescimento(periodoSel){

  if (periodoSel === "1s")  return "taxaCrescimento\_1semana";

  if (periodoSel === "1m")  return "taxaCrescimento\_1mes";

  return "taxaCrescimento\_1ano";

}

function melhorTaxaDisponivel(acao, prefer){

  const ordem = [prefer, "taxaCrescimento\_1mes", "taxaCrescimento\_1semana", "taxaCrescimento\_1ano"];

  for (const k of ordem){

    const v = Number(acao[k] || 0);

    if (v !== 0) return v;

  }

  return 0;

}

function calcularMetricasAcao(acao, periodoSel, horizonte){

  const prefer = campoCrescimento(periodoSel);

  const taxaPct = melhorTaxaDisponivel(acao, prefer);

  const preco     = Number(acao.valorStock || 0);

  const dividendo = Number(acao.dividendo || 0);

  const per       = acao.periodicidade || "Anual";

  if (!(preco>0)) return null;

  const r = clamp(taxaPct/100, -0.95, 5);

  const h = Math.max(1, Number(horizonte||1));

  const mult = Math.pow(1+r, h);

  const valorizacao = preco \* (mult - 1);

  const totalDividendos = dividirPeriodicidade(dividendo, per) \* h;

  const lucroUnidade = totalDividendos + valorizacao;

  const retornoPorEuro = lucroUnidade / preco;

  return { preco, dividendoAnual: dividirPeriodicidade(dividendo, per), taxaPct, lucroUnidade, retornoPorEuro };

}

function distribuirFracoes(acoes, investimento){

  const somaRetorno = acoes.reduce((s,a)=>s + a.metrics.retornoPorEuro, 0);

  if (somaRetorno <= 0) return { linhas: [], totalLucro: 0, totalGasto: 0, restante: investimento };

  const linhas = acoes.map(a=>{

    const propor = a.metrics.retornoPorEuro / somaRetorno;

    const investido = investimento \* propor;

    const qtd = investido / a.metrics.preco;

    const lucro = qtd \* a.metrics.lucroUnidade;

    return {

      nome: a.nome, ticker: a.ticker,

      preco: a.metrics.preco,

      quantidade: qtd,

      investido,

      lucro,

      taxaPct: a.metrics.taxaPct,

      dividendoAnual: a.metrics.dividendoAnual

    };

  });

  const totalLucro = linhas.reduce((s,l)=>s+l.lucro,0);

  const totalGasto = linhas.reduce((s,l)=>s+l.investido,0);

  return { linhas, totalLucro, totalGasto, restante: Math.max(0, investimento - totalGasto) };

}

function distribuirInteiras(acoes, investimento){

  const ordenadas = [...acoes].sort((a,b)=>b.metrics.retornoPorEuro - a.metrics.retornoPorEuro);

  const linhasMap = new Map();

  let restante = investimento;

  const precoMin = Math.min(...ordenadas.map(a=>a.metrics.preco));

  while (restante >= precoMin - 1e-9){

    let best = null;

    for (const a of ordenadas){

      if (a.metrics.preco <= restante + 1e-9){ best = a; break; }

    }

    if (!best) break;

    const key = best.ticker;

    const linha = linhasMap.get(key) || {

      nome: best.nome, ticker: best.ticker,

      preco: best.metrics.preco,

      quantidade: 0, investido: 0, lucro: 0,

      taxaPct: best.metrics.taxaPct,

      dividendoAnual: best.metrics.dividendoAnual

    };

    linha.quantidade += 1;

    linha.investido  += best.metrics.preco;

    linha.lucro      += best.metrics.lucroUnidade;

    linhasMap.set(key, linha);

    restante -= best.metrics.preco;

  }

  const linhas = Array.from(linhasMap.values());

  const totalLucro = linhas.reduce((s,l)=>s+l.lucro,0);

  const totalGasto = linhas.reduce((s,l)=>s+l.investido,0);

  return { linhas, totalLucro, totalGasto, restante };

}

function renderResultado(destEl, resultado, opts){

  const { linhas, totalLucro, totalGasto, restante=0 } = resultado;

  if (!linhas || linhas.length===0){

    destEl.innerHTML = `<div class="card"><p class="muted">Nenhuma ação selecionada com retorno positivo.</p></div>`;

    return;

  }

  const rows = linhas.map(l=>`

    <tr>

      <td>${l.nome} <span class="muted">(${l.ticker})</span></td>

      <td>${euro(l.preco)}</td>

      <td>${l.quantidade.toFixed( opts.inteiras ? 0 : 4 )}</td>

      <td>${euro(l.investido)}</td>

      <td>${euro(l.lucro)}</td>

      <td>${(l.taxaPct||0).toFixed(2)}%</td>

      <td>${euro(l.dividendoAnual||0)}/ano</td>

    </tr>

  `).join("");

  destEl.innerHTML = `

    <div class="card">

      <div class="tabela-scroll-wrapper">

        <table style="width:100%; border-collapse:collapse;">

          <thead>

            <tr>

              <th>Ativo</th><th>Preço</th><th>Qtd</th><th>Investido</th><th>Lucro Estim.</th><th>Tx ${opts.periodo}</th><th>Dividendo</th>

            </tr>

          </thead>

          <tbody>${rows}</tbody>

        </table>

      </div>

      <p style="margin-top:.6rem">

        <strong>Total investido:</strong> ${euro(totalGasto)}

        ${opts.inteiras && restante>0 ? `• <strong>Resto:</strong> ${euro(restante)}` : ``}

        <br/>

        <strong>Lucro total estimado (${opts.horizonte} ${opts.horizonte>1?"períodos":"período"}):</strong> ${euro(totalLucro)}

      </p>

    </div>

  `;

}

// --- opções dropdowns

const OPT\_SETORES = [

  "", "ETF iTech","ETF Finance","ETF Energia","ETF Materiais",

  "Alimentação","Automóvel","Bens Consumidor","Consumo Cíclico","Consumo Defensivo","Criptomoedas",

  "Defesa","Energia","Finanças","Imobiliário","Indústria",

  "Infraestruturas / Energia","Materiais","Mineração (Ouro)","Restauração","Saúde","Tecnologia","Telecomunicações"

];

const OPT\_MERCADOS = ["", "Portugal","Europeu","Americano","Americano SP500"];

const OPT\_MESES = ["", "n/A","Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro"];

const OPT\_PERIODICIDADE = ["", "n/A","Anual","Semestral","Trimestral","Mensal"];

function fillSelect(id, opts, placeholder){

  const el = document.getElementById(id); if(!el) return;

  el.innerHTML = opts.map(v=>{

    const label = v || (placeholder||"Todos");

    return `<option value="${v}">${label}</option>`;

  }).join("");

}

// ---------- Estado do modal ----------

let GOAL\_CACHE = [];

const GOAL\_SELECTED = new Map();

// fetch acoesDividendos (inclui setor/mercado/mes/periodicidade)

async function fetchAcoesGoal(){

  const snap = await getDocs(collection(db, "acoesDividendos"));

  const out = [];

  snap.forEach(doc=>{

    const d = doc.data();

    if (!d?.ticker) return;

    out.push({

      nome: d.nome || d.ticker,

      ticker: String(d.ticker).toUpperCase(),

      valorStock: Number(d.valorStock || 0),

      dividendo: Number(d.dividendo || 0),

      mes: d.mes || "",

      periodicidade: d.periodicidade || "Anual",

      taxaCrescimento\_1semana: Number(d.taxaCrescimento\_1semana || 0),

      taxaCrescimento\_1mes: Number(d.taxaCrescimento\_1mes || 0),

      taxaCrescimento\_1ano: Number(d.taxaCrescimento\_1ano || 0),

      raw: { setor: d.setor || "", mercado: d.mercado || "" }

    });

  });

  return out;

}

// ----- Wizard helpers -----

function showGoalStep(n){

  document.querySelectorAll('#goalModal .goal-step').forEach(sec=>{

    sec.hidden = sec.dataset.step !== String(n);

  });

}

function ensureListVisible(){

  const evt = new Event('click');

  document.getElementById('goalBtnFiltrar')?.dispatchEvent(evt);

}

// ----- Abrir/fechar -----

const btnObjetivo = document.getElementById("btnObjetivo");

const goalModal   = document.getElementById("goalModal");

const goalClose   = document.getElementById("goalClose");

btnObjetivo?.addEventListener("click", async () => {

  // always open at step 1

  showGoalStep(1);

  // preencher dropdowns

  fillSelect("goalFiltroSetor", OPT\_SETORES, "Setor");

  fillSelect("goalFiltroMercado", OPT\_MERCADOS, "Mercado");

  fillSelect("goalFiltroMes", OPT\_MESES, "Mês do dividendo");

  fillSelect("goalFiltroPeriodicidade", OPT\_PERIODICIDADE, "Periodicidade");

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

  if (GOAL\_CACHE.length === 0) GOAL\_CACHE = await fetchAcoesGoal();

  renderGoalList(GOAL\_CACHE);

  renderGoalSelected();

  syncGoalCheckboxes();

});

goalClose?.addEventListener("click", closeGoalModal);

goalModal?.addEventListener("click", (e) => {

  if (e.target.id === "goalModal") closeGoalModal();

});

function closeGoalModal() {

  goalModal?.classList.add("hidden");

  goalModal?.classList.remove("goal-show-results");

  GOAL\_SELECTED.clear();

  const box = document.getElementById("goalResultado");

  if (box) box.innerHTML = "";

  const bar = document.getElementById("goalResultsBar");

  if (bar) bar.remove();

  showGoalStep(1);

}

// ----- Passos -----

document.getElementById('goalNext1')?.addEventListener('click', ()=>{

  showGoalStep(2);

  ensureListVisible();

});

document.getElementById('goalNext2')?.addEventListener('click', ()=>{

  if (GOAL\_SELECTED.size === 0) { alert('Seleciona pelo menos uma ação.'); return; }

  showGoalStep(3);

});

document.getElementById('goalBack2')?.addEventListener('click', ()=> showGoalStep(1));

document.getElementById('goalBack3')?.addEventListener('click', ()=> showGoalStep(2));

document.getElementById('goalBackToEdit')?.addEventListener('click', ()=> showGoalStep(3));

// ----- Filtrar/Limpar -----

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

  const t   = (document.getElementById("goalFiltroTicker")?.value || "").trim().toLowerCase();

  const n   = (document.getElementById("goalFiltroNome")?.value   || "").trim().toLowerCase();

  const s   = (document.getElementById("goalFiltroSetor")?.value  || "").trim().toLowerCase();

  const m   = (document.getElementById("goalFiltroMercado")?.value|| "").trim().toLowerCase();

  const mes = (document.getElementById("goalFiltroMes")?.value    || "").trim().toLowerCase();

  const per = (document.getElementById("goalFiltroPeriodicidade")?.value||"").trim().toLowerCase();

  const res = GOAL\_CACHE.filter(a => {

    const hitT   = !t   || a.ticker.toLowerCase().includes(t);

    const hitN   = !n   || a.nome.toLowerCase().includes(n);

    const hitS   = !s   || String(a.raw?.setor||"").toLowerCase()      === s;

    const hitM   = !m   || String(a.raw?.mercado||"").toLowerCase()    === m;

    const hitMes = !mes || String(a.mes||"").toLowerCase()             === mes;

    const hitPer = !per || String(a.periodicidade||"").toLowerCase()   === per;

    return hitT && hitN && hitS && hitM && hitMes && hitPer;

  });

  renderGoalList(res);

});

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

  ["goalFiltroTicker","goalFiltroNome","goalFiltroSetor","goalFiltroMercado","goalFiltroMes","goalFiltroPeriodicidade"]

    .forEach(id=>{ const el = document.getElementById(id); if (el) el.value = ""; });

  renderGoalList(GOAL\_CACHE);

});

// ----- Lista (toggle por botão “＋/✓”) -----

function renderGoalList(arr){

  const ul = document.getElementById("goalListaAcoes");

  if (!ul) return;

  if (!arr || arr.length === 0) {

    ul.innerHTML = `<li><span class="meta">Sem resultados.</span></li>`;

    return;

  }

  ul.innerHTML = arr.map(a => {

    const selected = GOAL\_SELECTED.has(a.ticker);

    const setor   = a.raw?.setor || "-";

    const mercado = a.raw?.mercado || "-";

    const preco   = Number(a.valorStock||0).toFixed(2);

    return `

      <li>

        <div style="display:flex;align-items:center;gap:10px;flex:1;">

          <div>

            <div><strong>${a.nome}</strong> <span class="meta">(${a.ticker})</span></div>

            <div class="meta">${setor} • ${mercado} • Preço: €${preco}</div>

          </div>

        </div>

        <button class="icon-btn goal-toggle" data-ticker="${a.ticker}" title="${selected?'Remover':'Adicionar'}">

          ${selected ? '✓' : '＋'}

        </button>

      </li>

    `;

  }).join("");

  ul.querySelectorAll('.goal-toggle').forEach(btn=>{

    btn.addEventListener('click', () => {

      const t = btn.dataset.ticker;

      const obj = arr.find(x=>x.ticker===t) || GOAL\_CACHE.find(x=>x.ticker===t);

      if (!obj) return;

      if (GOAL\_SELECTED.has(t)) {

        GOAL\_SELECTED.delete(t);

        btn.textContent = '＋';

        btn.title = 'Adicionar';

      } else {

        GOAL\_SELECTED.set(t, obj);

        btn.textContent = '✓';

        btn.title = 'Remover';

      }

      renderGoalSelected();

    });

  });

}

// ----- Selecionadas (tags) -----

function renderGoalSelected(){

  const wrap = document.getElementById("goalSelecionadas");

  if (!wrap) return;

  const list = Array.from(GOAL\_SELECTED.values());

  if (list.length === 0) {

    wrap.innerHTML = `<span class="muted">Nenhuma ação selecionada.</span>`;

    return;

  }

  wrap.innerHTML = list.map(a => `

    <span class="goal-tag">

      ${a.ticker}

      <button title="Remover" data-del="${a.ticker}">x</button>

    </span>

  `).join("");

  wrap.querySelectorAll('button[data-del]').forEach(btn=>{

    btn.addEventListener("click", () => {

      const t = btn.dataset.del;

      GOAL\_SELECTED.delete(t);

      renderGoalSelected();

      const b = document.querySelector(`#goalListaAcoes .goal-toggle[data-ticker="${t}"]`);

      if (b){ b.textContent = '＋'; b.title = 'Adicionar'; }

    });

  });

}

// ----- Exclusividade das duas checkboxes -----

function syncGoalCheckboxes(){

  const chkInt = document.getElementById("goalAcoesCompletas");

  const chkTot = document.getElementById("goalUsarTotal");

  if (!chkInt || !chkTot) return;

  function onChange(e){

    if (e.target === chkInt && chkInt.checked) chkTot.checked = false;

    if (e.target === chkTot && chkTot.checked) chkInt.checked = false;

  }

  chkInt.onchange = onChange;

  chkTot.onchange = onChange;

  onChange({target: chkInt.checked ? chkInt : chkTot});

}

// ----- Simular -----

document.getElementById("goalBtnSimular")?.addEventListener("click", async ()=>{

  const invest   = Number(document.getElementById("goalInvest")?.value || 0);

  const periodo  = (document.getElementById("goalPeriodo")?.value || "1ano");

  const horizonte= Math.max(1, Number(document.getElementById("goalHorizonte")?.value || 1));

  const inteiras = !!document.getElementById("goalAcoesCompletas")?.checked;

  const usarTot  = !!document.getElementById("goalUsarTotal")?.checked;

  const box      = document.getElementById("goalResultado");

  if (box) box.innerHTML = `<div class="card">A simular…</div>`;

  const baseSelecionada = Array.from(GOAL\_SELECTED.values());

  if (baseSelecionada.length === 0){ box.innerHTML = `<div class="card"><p class="muted">Seleciona pelo menos uma ação.</p></div>`; return; }

  if (invest <= 0){ box.innerHTML = `<div class="card"><p class="muted">Indica o montante a investir.</p></div>`; return; }

  try{

    const comMetricas = baseSelecionada

      .map(a=>{ const m = calcularMetricasAcao(a, periodo, horizonte); return m ? {...a, metrics:m} : null; })

      .filter(Boolean)

      .filter(a=>a.metrics.retornoPorEuro > 0);

    if (comMetricas.length === 0){

      box.innerHTML = `<div class="card"><p class="muted">As ações selecionadas não têm retorno positivo para o período escolhido.</p></div>`;

      return;

    }

    const resultado = inteiras

      ? distribuirInteiras(comMetricas, invest)

      : distribuirFracoes(comMetricas, invest);

    renderResultado(box, resultado, { periodo, horizonte, inteiras, usarTot });

    goalEnterResultsMode();        // mostra modo resultado

  }catch(err){

    console.error(err);

    box.innerHTML = `<div class="card"><p class="muted">Ocorreu um erro na simulação.</p></div>`;

  }

});

// === UI: modo resultado ===

function goalEnterResultsMode() {

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

  if (!modal) return;

  showGoalStep(4);                           // step 4 = resultados

  modal.classList.add("goal-show-results");

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

    const bar = document.createElement("div");

    bar.id = "goalResultsBar";

    bar.className = "goal-results-bar";

    bar.innerHTML = `

      <button id="goalBackToEdit" class="icon-btn close" title="Voltar">×</button>

      <span class="title">Resultado da simulação</span>

    `;

    const res = document.getElementById("goalResultado");

    if (res && res.parentNode) res.parentNode.insertBefore(bar, res);

    document.getElementById("goalBackToEdit")?.addEventListener("click", goalExitResultsMode);

  }

}

function goalExitResultsMode() {

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

  if (!modal) return;

  modal.classList.remove("goal-show-results");

  showGoalStep(3);                            // regressa ao passo 3

}

//Compra acoes dashboard.js

// Abre o modal "Adicionar Ação" já em modo COMPRA e limpa os campos

function wireDashBuy() {

  const btn        = document.getElementById("btnDashBuy");

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

  const form       = document.getElementById("addForm");

  const tipoAcao   = document.getElementById("tipoAcao");

  const labelPreco = document.getElementById("labelPreco");

  const open = () => {

    if (!modal) return;

    modal.classList.remove("hidden");

    // força COMPRA

    if (tipoAcao) tipoAcao.value = "compra";

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

    // limpa campos do formulário

    ["nomeAtivo","tickerAtivo","setorAtivo","mercadoAtivo","quantidadeAtivo","precoAtivo","objetivoAtivo"]

      .forEach(id => { const el = document.getElementById(id); if (el) el.value = ""; });

  };

  const close = () => {

    if (!modal) return;

    modal.classList.add("hidden");

    form?.reset();

    if (labelPreco) labelPreco.firstChild.textContent = "Preço da transação (€)";

  };

  // abrir via botão principal

  btn?.addEventListener("click", open);

  // fechos já existentes do modal "Adicionar Ação"

  document.getElementById("addClose") ?.addEventListener("click", close);

  document.getElementById("addCancel")?.addEventListener("click", close);

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

  // se o utilizador mudar entre compra/venda, atualiza o label

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

    if (!labelPreco) return;

    labelPreco.firstChild.textContent =

      tipoAcao.value === "venda" ? "Preço de venda (€)" : "Preço de compra (€)";

  });

}