import React, { useState, useEffect, useCallback, useMemo, useRef } from 'react';

import {

BarChart, Bar, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer, PieChart, Pie, Cell, LineChart, Line

} from 'recharts';

// Firebase imports

import { initializeApp } from 'firebase/app';

import { getAuth, signInAnonymously, signInWithCustomToken, onAuthStateChanged } from 'firebase/auth';

import { getFirestore, doc, setDoc, onSnapshot, collection, query, where, getDocs } from 'firebase/firestore';

// --- UTILS E CONSTANTES ---

// Definição dos temas com paletas de cores

const themes = {

light: {

name: 'Claro',

background: '#F3F4F6', // gray-100

cardBackground: '#FFFFFF',

textColor: '#1F2937', // gray-900

secondaryTextColor: '#4B5563', // gray-600

primaryColor: '#3B82F6', // blue-500

secondaryColor: '#10B981', // green-500

accentColor: '#8B5CF6', // purple-500

dangerColor: '#EF4444', // red-500

warningColor: '#F59E0B', // yellow-500

chartPrimary: '#8884d8',

chartSecondary: '#82ca9d',

chartAccent: '#ff7300',

chartPriority: '#3f51b5',

chartRisk: '#e91e63',

tableHeaderBg: '#F9FAFB', // gray-50

tableBorder: '#E5E7EB', // gray-200

},

dark: {

name: 'Escuro',

background: '#1F2937', // gray-900

cardBackground: '#374151', // gray-700

textColor: '#F9FAFB', // gray-50

secondaryTextColor: '#D1D5DB', // gray-300

primaryColor: '#60A5FA', // blue-400

secondaryColor: '#34D399', // green-400

accentColor: '#A78BFA', // purple-400

dangerColor: '#F87171', // red-400

warningColor: '#FBBF24', // yellow-400

chartPrimary: '#A78BFA', // purple-400

chartSecondary: '#34D399', // green-400

chartAccent: '#FBBF24', // yellow-400

chartPriority: '#60A5FA', // blue-400

chartRisk: '#F87171', // red-400

tableHeaderBg: '#4B5563', // gray-600

tableBorder: '#6B7280', // gray-500

},

'blue-green': {

name: 'Azul-Verde',

background: '#E0F2F7', // light blue

cardBackground: '#FFFFFF',

textColor: '#2C3E50', // dark blue-gray

secondaryTextColor: '#5D6D7E',

primaryColor: '#3498DB', // strong blue

secondaryColor: '#2ECC71', // strong green

accentColor: '#9B59B6', // purple

dangerColor: '#E74C3C', // red

warningColor: '#F39C12', // orange

chartPrimary: '#3498DB',

chartSecondary: '#2ECC71',

chartAccent: '#F39C12',

chartPriority: '#34495E',

chartRisk: '#E74C3C',

tableHeaderBg: '#ECF0F1',

tableBorder: '#BDC3C7',

}

};

// Função auxiliar para calcular a luminância de uma cor hexadecimal

const getLuminance = (hexColor) => {

const hex = hexColor.substring(1); // Remove #

const r = parseInt(hex.substring(0, 2), 16);

const g = parseInt(hex.substring(2, 4), 16);

const b = parseInt(hex.substring(4, 6), 16);

return (0.299 \* r + 0.587 \* g + 0.114 \* b) / 255;

};

// Formata data

const formatDate = (dateString) => {

if (!dateString) return 'N/A';

try {

const date = new Date(dateString);

return date.toLocaleDateString('pt-BR');

} catch (e) {

return dateString;

}

};

// Formata moeda

const formatCurrency = (value) => {

if (value === undefined || value === null) return 'N/A';

return new Intl.NumberFormat('pt-BR', { style: 'currency', currency: 'BRL' }).format(value);

};

// Cores padrão para gráficos de pizza

const PIE\_COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042', '#A28DFF', '#FF6384'];

// Definições de colunas para a tabela (usadas para visibilidade e exportação)

const columnDefinitions = [

{ key: 'atividade', label: 'Atividade', defaultVisible: true, type: 'text' },

{ key: 'disciplina', label: 'Disciplina', defaultVisible: true, type: 'text' },

{ key: 'responsavel', label: 'Resp.', defaultVisible: true, type: 'text' },

{ key: 'prioridade', label: 'Prioridade', defaultVisible: false, type: 'text' }, // Hidden by default

{ key: 'dataInicialPlanejada', label: 'Data Inic. Plan.', defaultVisible: false, type: 'date' }, // Hidden by default

{ key: 'dataFinalPlanejada', label: 'Data Fin. Plan.', defaultVisible: false, type: 'date' }, // Hidden by default

{ key: 'dataInicialReal', label: 'Data Inic. Real', defaultVisible: false, type: 'date' }, // Hidden by default

{ key: 'dataFinalReal', label: 'Data Fin. Real', defaultVisible: false, type: 'date' }, // Hidden by default

{ key: 'dataInicialBaseline', label: 'Data Inic. Baseline', defaultVisible: false, type: 'date' }, // New: Baseline start date

{ key: 'dataFinalBaseline', label: 'Data Fin. Baseline', defaultVisible: false, type: 'date' }, // New: Baseline end date

{ key: 'valorPlanejado', label: 'Valor Plan. (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'valorReal', label: 'Valor Real (R$)', defaultVisible: true, type: 'number' }, // Visible by default

{ key: 'custoReal', label: 'Custo Real (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'statusPlanejado', label: 'Status Plan.', defaultVisible: false, type: 'text' }, // Hidden by default

{ key: 'statusReal', label: 'Status Real', defaultVisible: true, type: 'text' }, // Visible by default

{ key: 'percentualConclusao', label: 'Conclusão (%)', defaultVisible: true, type: 'number' }, // Visible by default

{ key: 'riscoAssociado', label: 'Risco', defaultVisible: false, type: 'text' }, // Hidden by default

{ key: 'diasAtraso', label: 'Dias Atraso', defaultVisible: true, type: 'number' }, // Visible by default

{ key: 'diasRemanescentes', label: 'Dias Remanescentes', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'recursosNecessarios', label: 'Recursos Necessários', defaultVisible: false, type: 'text' }, // Hidden by default

{ key: 'linkDocumento', label: 'Link/Documento', defaultVisible: false, type: 'text' }, // Hidden by default

{ key: 'dataUltimaAtualizacao', label: 'Última Atualização', defaultVisible: false, type: 'date' }, // Hidden by default

// Novos campos para KPIs de Valor Agregado

{ key: 'orcamentoPlanejado', label: 'Orçamento Plan. (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'valorAgregado', label: 'Valor Agregado (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'variacaoPrazo', label: 'Variação Prazo (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'indiceDesempenhoPrazo', label: 'SPI', defaultVisible: true, type: 'number' }, // Visible by default

{ key: 'variacaoCusto', label: 'Variação Custo (R$)', defaultVisible: false, type: 'number' }, // Hidden by default

{ key: 'indiceDesempenhoCusto', label: 'CPI', defaultVisible: true, type: 'number' }, // Visible by default

{ key: 'actions', label: 'Ações', defaultVisible: true, type: 'action' },

];

// Função para avaliar fórmulas de forma segura

const evaluateFormula = (activity, formula, allAvailableColumns) => {

if (!formula || typeof formula !== 'string') return 'N/A';

let evaluatedFormula = formula;

// Replace column names with their numeric values

allAvailableColumns.forEach(col => {

const value = parseFloat(activity[col.key]) || 0;

// Use a regex to replace the whole word to avoid partial matches

evaluatedFormula = evaluatedFormula.replace(new RegExp(`\\b${col.key}\\b`, 'g'), value);

});

try {

// Basic arithmetic evaluation. This is still somewhat risky if formula contains non-numeric operations.

// For production, a more robust and secure math expression parser would be needed.

// For this context, assuming simple arithmetic with numbers.

const result = new Function('return ' + evaluatedFormula)();

return isNaN(result) ? 'Erro' : result;

} catch (e) {

console.error("Erro ao avaliar fórmula:", formula, e);

return 'Erro';

}

};

// --- HOOKS CUSTOMIZADOS ---

/\*\*

\* Hook para gerenciar a autenticação Firebase e os dados do dashboard.

\*/

const useDashboardData = () => {

const appId = typeof \_\_app\_id !== 'undefined' ? \_\_app\_id : 'default-app-id';

const firebaseConfig = useMemo(() => JSON.parse(typeof \_\_firebase\_config !== 'undefined' ? \_\_firebase\_config : '{}'), []);

const [db, setDb] = useState(null);

const [auth, setAuth] = useState(null);

const [userId, setUserId] = useState(null);

const [isAuthReady, setIsAuthReady] = useState(false);

const [activeDashboardId, setActiveDashboardId] = useState(null);

const [activeDashboardName, setActiveDashboardName] = useState('');

const [allActivities, setAllActivities] = useState([]);

const [customStatuses, setCustomStatuses] = useState([]);

const [customRisks, setCustomRisks] = useState([]);

const [customColumns, setCustomColumns] = useState([]); // Novo estado para colunas personalizadas

const [customCharts, setCustomCharts] = useState([]); // Novo estado para gráficos personalizados

// Inicializa Firebase e configura o listener de autenticação

useEffect(() => {

try {

const app = initializeApp(firebaseConfig);

const firestore = getFirestore(app);

const firebaseAuth = getAuth(app);

setDb(firestore);

setAuth(firebaseAuth);

const unsubscribe = onAuthStateChanged(firebaseAuth, async (user) => {

if (user) {

setUserId(user.uid);

} else {

try {

if (typeof \_\_initial\_auth\_token !== 'undefined') {

await signInWithCustomToken(firebaseAuth, \_\_initial\_auth\_token);

} else {

await signInAnonymously(firebaseAuth);

}

setUserId(firebaseAuth.currentUser?.uid || crypto.randomUUID());

} catch (anonError) {

console.error("Erro ao autenticar anonimamente:", anonError);

}

}

setIsAuthReady(true);

});

return () => unsubscribe();

} catch (err) {

console.error("Erro ao inicializar Firebase:", err);

}

}, [firebaseConfig]);

// Carrega dados do dashboard quando o ID do dashboard ativo muda

useEffect(() => {

if (db && activeDashboardId && isAuthReady) {

const dashboardDocRef = doc(db, `artifacts/${appId}/public/data/dashboards`, activeDashboardId);

const unsubscribe = onSnapshot(dashboardDocRef, (docSnap) => {

if (docSnap.exists()) {

const data = docSnap.data();

setAllActivities(data.activities || []);

setCustomStatuses(data.customStatuses || []);

setCustomRisks(data.customRisks || []);

setCustomColumns(data.customColumns || []); // Carrega colunas personalizadas

setCustomCharts(data.customCharts || []); // Carrega gráficos personalizados

setActiveDashboardName(data.name || 'Dashboard Sem Nome');

} else {

console.log("Dashboard não encontrado!");

setAllActivities([]);

setCustomStatuses([]);

setCustomRisks([]);

setCustomColumns([]);

setCustomCharts([]);

setActiveDashboardId(null);

setActiveDashboardName('');

}

}, (err) => {

console.error("Erro ao carregar dados do dashboard:", err);

});

return () => unsubscribe();

}

}, [db, activeDashboardId, isAuthReady, appId]);

// Salva atividades e configurações no Firestore com debounce

useEffect(() => {

if (db && activeDashboardId && isAuthReady) {

const dashboardDocRef = doc(db, `artifacts/${appId}/public/data/dashboards`, activeDashboardId);

const timeoutId = setTimeout(async () => {

try {

await setDoc(dashboardDocRef, { activities: allActivities, customStatuses, customRisks, customColumns, customCharts }, { merge: true });

} catch (e) {

console.error("Erro ao salvar dados no Firestore:", e);

}

}, 500);

return () => clearTimeout(timeoutId);

}

}, [allActivities, customStatuses, customRisks, customCharts, db, activeDashboardId, isAuthReady, appId, customColumns]);

return {

allActivities, setAllActivities, customStatuses, setCustomStatuses, customRisks, setCustomRisks,

customColumns, setCustomColumns, customCharts, setCustomCharts,

db, auth, userId, isAuthReady, activeDashboardId, setActiveDashboardId, activeDashboardName, setActiveDashboardName

};

};

/\*\*

\* Hook para gerenciar filtros e métricas calculadas das atividades.

\*/

const useActivityMetrics = (allActivities, customStatuses, customRisks, customColumns, searchTerm, filterStatus, filterResponsavel, startDate, endDate) => {

const allAvailableColumns = useMemo(() => [

...columnDefinitions.filter(col => col.type !== 'action'),

...customColumns.map(col => ({ key: col.id, label: col.name, type: col.type, formula: col.formula }))

], [customColumns]);

// Filtra atividades com base nos critérios de busca e data

const filteredActivities = useMemo(() => {

return allActivities.filter(activity => {

const activityDate = activity.dataInicialReal || activity.dataInicialPlanejada;

const matchesSearch = searchTerm === '' ||

Object.values(activity).some(value =>

String(value).toLowerCase().includes(searchTerm.toLowerCase())

);

const matchesStatus = filterStatus === 'Todos' || activity.statusReal === filterStatus;

const matchesResponsavel = filterResponsavel === 'Todos' || activity.responsavel === filterResponsavel;

let matchesDate = true;

if (startDate && endDate) {

if (activityDate) {

const start = new Date(startDate);

const end = new Date(endDate);

const currentActivityDate = new Date(activityDate);

matchesDate = currentActivityDate >= start && currentActivityDate <= end;

} else {

matchesDate = false;

}

}

return matchesSearch && matchesStatus && matchesResponsavel && matchesDate;

});

}, [allActivities, searchTerm, filterStatus, filterResponsavel, startDate, endDate]);

// Calcula dias de atraso e remanescentes, e avalia colunas de fórmula

const activitiesWithCalculatedMetrics = useMemo(() => {

const today = new Date();

today.setHours(0, 0, 0, 0);

return filteredActivities.map(activity => {

const plannedEndDate = activity.dataFinalPlanejada ? new Date(activity.dataFinalPlanejada) : null;

const realEndDate = activity.dataFinalReal ? new Date(activity.dataFinalReal) : null;

let diasAtraso = 0;

let diasRemanescentes = 0;

if (activity.statusReal === 'Atrasado' && plannedEndDate) {

diasAtraso = Math.max(0, Math.ceil((today.getTime() - plannedEndDate.getTime()) / (1000 \* 60 \* 60 \* 24)));

} else if (activity.statusReal === 'Concluído' && plannedEndDate && realEndDate && realEndDate > plannedEndDate) {

diasAtraso = Math.ceil((realEndDate.getTime() - plannedEndDate.getTime()) / (1000 \* 60 \* 60 \* 24));

} else if ((activity.statusReal === 'Em Andamento' || activity.statusReal === 'Não Iniciado') && plannedEndDate && today > plannedEndDate) {

diasAtraso = Math.ceil((today.getTime() - plannedEndDate.getTime()) / (1000 \* 60 \* 60 \* 24));

}

if (activity.statusReal !== 'Concluído' && plannedEndDate && plannedEndDate > today) {

diasRemanescentes = Math.ceil((plannedEndDate.getTime() - today.getTime()) / (1000 \* 60 \* 60 \* 24));

}

// Calculate Earned Value (EV)

const valorAgregado = (activity.percentualConclusao / 100) \* (activity.orcamentoPlanejado || activity.valorPlanejado || 0);

// Calculate Schedule Variance (SV) and Schedule Performance Index (SPI)

const variacaoPrazo = valorAgregado - (activity.orcamentoPlanejado || activity.valorPlanejado || 0);

const indiceDesempenhoPrazo = (activity.orcamentoPlanejado || activity.valorPlanejado || 0) !== 0

? valorAgregado / (activity.orcamentoPlanejado || activity.valorPlanejado || 0)

: 0;

// Calculate Cost Variance (CV) and Cost Performance Index (CPI)

const variacaoCusto = valorAgregado - (activity.valorReal || activity.custoReal || 0);

const indiceDesempenhoCusto = (activity.valorReal || activity.custoReal || 0) !== 0

? valorAgregado / (activity.valorReal || activity.custoReal || 0)

: 0;

// Avaliar colunas de fórmula

const calculatedCustomColumns = {};

customColumns.filter(col => col.type === 'formula').forEach(col => {

calculatedCustomColumns[col.id] = evaluateFormula(activity, col.formula, allAvailableColumns);

});

return {

...activity,

diasAtraso,

diasRemanescentes,

valorAgregado, // Add EV

variacaoPrazo, // Add SV

indiceDesempenhoPrazo, // Add SPI

variacaoCusto, // Add CV

indiceDesempenhoCusto, // Add CPI

...calculatedCustomColumns,

};

});

}, [filteredActivities, customColumns, allAvailableColumns]);

// Resumo de Status

const getStatusSummary = useCallback((type, activitiesToSummarize) => {

const summary = {

'Concluído': 0,

'Em Andamento': 0,

'Atrasado': 0,

'Não Iniciado': 0,

};

customStatuses.forEach(status => {

summary[status.name] = 0;

});

activitiesToSummarize.forEach(activity => {

const status = type === 'planejado' ? activity.statusPlanejado : activity.statusReal;

if (summary[status] !== undefined) {

summary[status]++;

}

});

return summary;

}, [customStatuses]);

const plannedSummary = useMemo(() => getStatusSummary('planejado', activitiesWithCalculatedMetrics), [getStatusSummary, activitiesWithCalculatedMetrics]);

const realSummary = useMemo(() => getStatusSummary('real', activitiesWithCalculatedMetrics), [getStatusSummary, activitiesWithCalculatedMetrics]);

const totalActivities = activitiesWithCalculatedMetrics.length;

const completedActivities = activitiesWithCalculatedMetrics.filter(act => act.statusReal === 'Concluído').length;

const overallCompletionPercentage = totalActivities > 0 ? (completedActivities / totalActivities) \* 100 : 0;

const uniqueResponsaveis = useMemo(() => [...new Set(allActivities.map(act => act.responsavel))], [allActivities]);

// Dados para gráficos

const statusChartData = useMemo(() => Object.keys(plannedSummary).map(status => ({

status: status,

Planejado: plannedSummary[status],

Real: realSummary[status],

})), [plannedSummary, realSummary]);

const deadlineDeviationData = useMemo(() => activitiesWithCalculatedMetrics.map(activity => {

const plannedEndDate = activity.dataFinalPlanejada ? new Date(activity.dataFinalPlanejada) : null;

const realEndDate = activity.dataFinalReal ? new Date(activity.dataFinalReal) : null;

let deviation = 0;

if (plannedEndDate && realEndDate) {

deviation = (realEndDate.getTime() - plannedEndDate.getTime()) / (1000 \* 60 \* 60 \* 24);

} else if (plannedEndDate && !realEndDate) {

deviation = (new Date().getTime() - plannedEndDate.getTime()) / (1000 \* 60 \* 60 \* 24);

}

return {

atividade: activity.atividade,

desvioDias: parseFloat(deviation.toFixed(1)),

};

}).sort((a, b) => b.desvioDias - a.desvioDias), [activitiesWithCalculatedMetrics]);

const costByDiscipline = useMemo(() => activitiesWithCalculatedMetrics.reduce((acc, activity) => {

if (!acc[activity.disciplina]) {

acc[activity.disciplina] = {

disciplina: activity.disciplina,

Planejado: 0,

Real: 0,

};

}

acc[activity.disciplina].Planejado += activity.valorPlanejado || 0;

acc[activity.disciplina].Real += activity.valorReal || 0;

return acc;

}, {}), [activitiesWithCalculatedMetrics]);

const costChartData = useMemo(() => Object.values(costByDiscipline), [costByDiscipline]);

const completionByDiscipline = useMemo(() => activitiesWithCalculatedMetrics.reduce((acc, activity) => {

if (!acc[activity.disciplina]) {

acc[activity.disciplina] = { total: 0, completed: 0 };

}

acc[activity.disciplina].total++;

if (activity.statusReal === 'Concluído') {

acc[activity.disciplina].completed++;

}

return acc;

}, {}), [activitiesWithCalculatedMetrics]);

const completionPieData = useMemo(() => Object.keys(completionByDiscipline).map(discipline => {

const total = completionByDiscipline[discipline].total;

const completed = completionByDiscipline[discipline].completed;

return {

name: discipline,

value: total > 0 ? (completed / total) \* 100 : 0,

};

}), [completionByDiscipline]);

const priorityData = useMemo(() => activitiesWithCalculatedMetrics.reduce((acc, activity) => {

acc[activity.prioridade] = (acc[activity.prioridade] || 0) + 1;

return acc;

}, {}), [activitiesWithCalculatedMetrics]);

const priorityChartData = useMemo(() => Object.keys(priorityData).map(priority => ({

name: priority,

count: priorityData[priority]

})), [priorityData]);

const riskData = useMemo(() => activitiesWithCalculatedMetrics.reduce((acc, activity) => {

acc[activity.riscoAssociado] = (acc[activity.riscoAssociado] || 0) + 1;

return acc;

}, {}), [activitiesWithCalculatedMetrics]);

const riskChartData = useMemo(() => Object.keys(riskData).map(risk => ({

name: risk,

count: riskData[risk]

})), [riskData]);

const heatmapData = useMemo(() => {

const data = {};

const allDisciplines = [...new Set(activitiesWithCalculatedMetrics.map(act => act.disciplina))];

const allResponsaveis = [...new Set(activitiesWithCalculatedMetrics.map(act => act.responsavel))];

allDisciplines.forEach(discipline => {

data[discipline] = {};

allResponsaveis.forEach(responsible => {

data[discipline][responsible] = 0;

});

});

activitiesWithCalculatedMetrics.forEach(activity => {

if (data[activity.disciplina] && data[activity.disciplina][activity.responsavel] !== undefined) {

data[activity.disciplina][activity.responsavel]++;

}

});

return data;

}, [activitiesWithCalculatedMetrics]);

const maxActivityCount = useMemo(() => {

let maxCount = 0;

Object.values(heatmapData).forEach(disciplineCounts => {

Object.values(disciplineCounts).forEach(count => {

if (count > maxCount) {

maxCount = count;

}

});

});

return maxCount;

}, [heatmapData]);

const allDisciplinesForHeatmap = useMemo(() => [...new Set(activitiesWithCalculatedMetrics.map(act => act.disciplina))], [activitiesWithCalculatedMetrics]);

const allResponsaveisForHeatmap = useMemo(() => [...new Set(activitiesWithCalculatedMetrics.map(act => act.responsavel))], [activitiesWithCalculatedMetrics]);

const overdueActivities = useMemo(() => activitiesWithCalculatedMetrics.filter(activity =>

activity.statusReal === 'Atrasado'

).sort((a, b) => {

const dateA = new Date(a.dataFinalPlanejada);

const dateB = new Date(b.dataFinalPlanejada);

return dateA - dateB;

}), [activitiesWithCalculatedMetrics]);

const remainingDaysActivities = useMemo(() => activitiesWithCalculatedMetrics.filter(activity =>

activity.diasRemanescentes > 0 && activity.statusReal !== 'Concluído'

).sort((a, b) => a.diasRemanescentes - b.diasRemanescentes), [activitiesWithCalculatedMetrics]);

const activitiesByResponsibleSummary = useMemo(() => activitiesWithCalculatedMetrics.reduce((acc, activity) => {

if (!acc[activity.responsavel]) {

acc[activity.responsavel] = {

total: 0,

concluido: 0,

emAndamento: 0,

atrasado: 0,

naoIniciado: 0,

};

}

acc[activity.responsavel].total++;

switch (activity.statusReal) {

case 'Concluído':

acc[activity.responsavel].concluido++;

break;

case 'Em Andamento':

acc[activity.responsavel].emAndamento++;

break;

case 'Atrasado':

acc[activity.responsavel].atrasado++;

break;

case 'Não Iniciado':

acc[activity.responsavel].naoIniciado++;

break;

default:

acc[activity.responsavel].emAndamento++;

break;

}

return acc;

}, {}), [activitiesWithCalculatedMetrics]);

const totalPlannedCost = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + act.valorPlanejado, 0), [activitiesWithCalculatedMetrics]);

const totalRealCost = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + act.valorReal, 0), [activitiesWithCalculatedMetrics]);

const costDeviation = useMemo(() => totalRealCost - totalPlannedCost, [totalRealCost, totalPlannedCost]);

const totalDiasAtraso = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + act.diasAtraso, 0), [activitiesWithCalculatedMetrics]);

const totalDiasRemanescentes = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + act.diasRemanescentes, 0), [activitiesWithCalculatedMetrics]);

// New KPIs for Earned Value Management (EVM)

const totalOrcamentoPlanejado = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + (act.orcamentoPlanejado || act.valorPlanejado || 0), 0), [activitiesWithCalculatedMetrics]);

const totalValorAgregado = useMemo(() => activitiesWithCalculatedMetrics.reduce((sum, act) => sum + act.valorAgregado, 0), [activitiesWithCalculatedMetrics]);

const overallVariacaoPrazo = totalValorAgregado - totalOrcamentoPlanejado;

const overallIndiceDesempenhoPrazo = totalOrcamentoPlanejado !== 0 ? totalValorAgregado / totalOrcamentoPlanejado : 0;

const overallVariacaoCusto = totalValorAgregado - totalRealCost;

const overallIndiceDesempenhoCusto = totalRealCost !== 0 ? totalValorAgregado / totalRealCost : 0;

// Novas métricas para gráficos cruzados

const allPossibleStatuses = useMemo(() => [...new Set(['Concluído', 'Em Andamento', 'Atrasado', 'Não Iniciado', ...customStatuses.map(s => s.name)])], [customStatuses]);

const disciplineStatusData = useMemo(() => {

const data = {};

activitiesWithCalculatedMetrics.forEach(activity => {

if (!data[activity.disciplina]) {

data[activity.disciplina] = { disciplina: activity.disciplina };

allPossibleStatuses.forEach(status => {

data[activity.disciplina][status] = 0;

});

}

if (data[activity.disciplina][activity.statusReal] !== undefined) {

data[activity.disciplina][activity.statusReal]++;

}

});

return Object.values(data);

}, [activitiesWithCalculatedMetrics, allPossibleStatuses]);

const responsibleStatusData = useMemo(() => {

const data = {};

activitiesWithCalculatedMetrics.forEach(activity => {

if (!data[activity.responsavel]) {

data[activity.responsavel] = { responsavel: activity.responsavel };

allPossibleStatuses.forEach(status => {

data[activity.responsavel][status] = 0;

});

}

if (data[activity.responsavel][activity.statusReal] !== undefined) {

data[activity.responsavel][activity.statusReal]++;

}

});

return Object.values(data);

}, [activitiesWithCalculatedMetrics, allPossibleStatuses]);

return {

activitiesWithCalculatedMetrics,

plannedSummary,

realSummary,

totalActivities,

completedActivities,

overallCompletionPercentage,

uniqueResponsaveis,

statusChartData,

deadlineDeviationData,

costByDiscipline,

costChartData,

completionByDiscipline,

completionPieData,

priorityData,

priorityChartData,

riskData,

riskChartData,

heatmapData,

maxActivityCount,

allDisciplinesForHeatmap,

allResponsaveisForHeatmap,

overdueActivities,

remainingDaysActivities,

activitiesByResponsibleSummary,

totalPlannedCost,

totalRealCost,

costDeviation,

totalDiasAtraso,

totalDiasRemanescentes,

disciplineStatusData, // Novo

responsibleStatusData, // Novo

allPossibleStatuses, // Novo

allAvailableColumns, // Novo: para seleção de colunas em gráficos e fórmulas

// New EVM KPIs

totalOrcamentoPlanejado,

totalValorAgregado,

overallVariacaoPrazo,

overallIndiceDesempenhoPrazo,

overallVariacaoCusto,

overallIndiceDesempenhoCusto,

};

};

/\*\*

\* Hook para gerenciar a funcionalidade de importação e exportação de CSV.

\*/

const useCsvImportExport = (setAllActivities, showFeedback, activitiesWithCalculatedMetrics, columnDefinitions, customColumns) => {

const [csvFile, setCsvFile] = useState(null);

const [isImportingCsv, setIsImportingCsv] = useState(false);

const [csvPreviewData, setCsvPreviewData] = useState([]);

const [csvPreviewHeaders, setCsvPreviewHeaders] = useState([]);

const [showCsvPreviewModal, setShowCsvPreviewModal] = useState(false);

// Combina as definições de colunas padrão e personalizadas para exportação

const allExportableColumns = useMemo(() => [

...columnDefinitions.filter(col => col.key !== 'actions'),

...customColumns.map(col => ({ key: col.id, label: col.name, type: col.type })) // Usar id como key para custom columns

], [columnDefinitions, customColumns]);

// Função para parsear CSV

const parseCsv = useCallback((csvText) => {

const lines = csvText.split('\n').filter(line => line.trim() !== '');

if (lines.length === 0) {

throw new Error("Arquivo CSV vazio.");

}

const headers = lines[0].split(',').map(header => header.trim());

const data = [];

for (let i = 1; i < lines.length; i++) {

const values = lines[i].split(',');

if (values.length !== headers.length) {

console.warn(`Linha ${i + 1} ignorada devido a número inconsistente de colunas.`);

continue;

}

const row = {};

headers.forEach((header, index) => {

row[header] = values[index].trim();

});

data.push(row);

}

return { headers, data };

}, []);

// Lida com a seleção do arquivo CSV

const handleFileChange = useCallback((event) => {

const file = event.target.files[0];

if (!file) {

setCsvFile(null);

setCsvPreviewData([]);

setCsvPreviewHeaders([]);

setShowCsvPreviewModal(false);

return;

}

setCsvFile(file);

setIsImportingCsv(true);

const reader = new FileReader();

reader.onload = (e) => {

const text = e.target.result;

try {

const { headers, data } = parseCsv(text);

setCsvPreviewHeaders(headers);

setCsvPreviewData(data);

setShowCsvPreviewModal(true);

} catch (error) {

console.error("Erro ao processar CSV para pré-visualização:", error);

showFeedback(`Erro ao ler o arquivo CSV para pré-visualização: ${error.message}.`);

} finally {

setIsImportingCsv(false);

}

};

reader.onerror = () => {

showFeedback('Erro ao ler o arquivo CSV.');

setIsImportingCsv(false);

setCsvFile(null);

};

reader.readAsText(file);

}, [parseCsv, showFeedback]);

// Confirma a importação do CSV

const handleConfirmImport = useCallback(() => {

const newActivities = csvPreviewData.map(activity => {

const newActivity = {

id: Date.now() + Math.random(),

dependencias: activity.dependencias ? String(activity.dependencias).split(',').map(d => d.trim()) : [],

recursosNecessarios: activity.recursosNecessarios ? String(activity.recursosNecessarios).split(',').map(r => r.trim()) : [],

valorPlanejado: parseFloat(activity.valorPlanejado) || 0,

valorReal: parseFloat(activity.valorReal) || 0,

custoReal: parseFloat(activity.custoReal) || 0,

percentualConclusao: parseFloat(activity.percentualConclusao) || 0,

atividade: activity.atividade || 'Nova Atividade',

disciplina: activity.disciplina || 'Geral',

statusPlanejado: activity.statusPlanejado || 'Não Iniciado',

statusReal: activity.statusReal || 'Não Iniciado',

responsavel: activity.responsavel || 'Desconhecido',

prioridade: activity.prioridade || 'Média',

observacoes: activity.observacoes || '',

riscoAssociado: activity.riscoAssociado || 'Baixo',

linkDocumento: activity.linkDocumento || '',

dataUltimaAtualizacao: new Date().toISOString().split('T')[0],

// New EVM fields

orcamentoPlanejado: parseFloat(activity.orcamentoPlanejado) || 0,

// New Baseline fields for Gantt

dataInicialBaseline: activity.dataInicialPlanejada || '', // Default to planned date on import

dataFinalBaseline: activity.dataFinalPlanejada || '', // Default to planned date on import

};

// Adiciona dados de colunas personalizadas

customColumns.forEach(col => {

if (activity[col.name]) { // CSV might use column name, not ID

let value = activity[col.name];

if (col.type === 'number') {

newActivity[col.id] = parseFloat(value) || 0;

} else if (col.type === 'date') {

newActivity[col.id] = value; // Keep as string for date input

} else {

newActivity[col.id] = value;

}

} else {

// Initialize custom column if not present in CSV

newActivity[col.id] = col.type === 'number' ? 0 : (col.type === 'date' ? '' : '');

}

});

return newActivity;

});

setAllActivities(prev => [...prev, ...newActivities]);

showFeedback(`Importação de CSV concluída! ${newActivities.length} atividades adicionadas.`);

setShowCsvPreviewModal(false);

setCsvFile(null);

setCsvPreviewData([]);

setCsvPreviewHeaders([]);

}, [csvPreviewData, setAllActivities, showFeedback, customColumns]);

// Cancela a importação do CSV

const handleCancelImport = useCallback(() => {

setShowCsvPreviewModal(false);

setCsvFile(null);

setCsvPreviewData([]);

setCsvPreviewHeaders([]);

showFeedback('Importação de CSV cancelada.');

}, [showFeedback]);

// Função para escapar valores CSV

const escapeCsvValue = useCallback((value) => {

if (value === null || value === undefined) return '';

let stringValue = String(value);

if (stringValue.includes(',') || stringValue.includes('"') || stringValue.includes('\n')) {

return `"${stringValue.replace(/"/g, '""')}"`;

}

return stringValue;

}, []);

// Exporta dados da tabela para CSV

const handleExportCsv = useCallback(() => {

if (activitiesWithCalculatedMetrics.length === 0) {

showFeedback("Não há dados para exportar para CSV.");

return;

}

const csvHeaders = allExportableColumns.map(col => escapeCsvValue(col.label)).join(',');

const csvRows = activitiesWithCalculatedMetrics.map(activity => {

return allExportableColumns.map(col => {

let value = activity[col.key];

if (col.type === 'date' && value) {

value = formatDate(value);

} else if (col.type === 'number') {

value = parseFloat(value) || 0;

} else if (Array.isArray(value)) {

value = value.join('; ');

} else if (col.key === 'percentualConclusao') {

value = `${value}%`;

}

return escapeCsvValue(value);

}).join(',');

});

const csvContent = [csvHeaders, ...csvRows].join('\n');

const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });

const link = document.createElement('a');

const url = URL.createObjectURL(blob);

link.setAttribute('href', url);

link.setAttribute('download', 'atividades.csv');

link.style.visibility = 'hidden';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

URL.revokeObjectURL(url);

showFeedback("Tabela exportada para atividades.csv!");

}, [activitiesWithCalculatedMetrics, allExportableColumns, escapeCsvValue, showFeedback]);

return {

csvFile, isImportingCsv, csvPreviewData, csvPreviewHeaders, showCsvPreviewModal,

handleFileChange, handleConfirmImport, handleCancelImport, handleExportCsv

};

};

/\*\*

\* Hook para gerenciar a geração de relatórios PDF.

\*/

const usePdfReport = (showFeedback) => {

const [isGeneratingPdf, setIsGeneratingPdf] = useState(false);

const [pdfLibsLoaded, setPdfLibsLoaded] = useState(false);

// Efeito para carregar as bibliotecas html2canvas e jspdf dinamicamente

useEffect(() => {

const loadScript = (src, id, onloadCallback) => {

if (document.getElementById(id)) {

onloadCallback();

return;

}

const script = document.createElement('script');

script.src = src;

script.id = id;

script.onload = onloadCallback;

script.onerror = () => console.error(`Falha ao carregar o script: ${src}`);

document.body.appendChild(script);

};

let html2canvasLoaded = false;

let jspdfLoaded = false;

const checkAllLoaded = () => {

if (html2canvasLoaded && jspdfLoaded) {

setPdfLibsLoaded(true);

}

};

loadScript(

"https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js",

"html2canvas-script",

() => {

html2canvasLoaded = true;

checkAllLoaded();

}

);

loadScript(

"https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js",

"jspdf-script",

() => {

jspdfLoaded = true;

checkAllLoaded();

}

);

return () => {

const html2canvasScript = document.getElementById("html2canvas-script");

const jspdfScript = document.getElementById("jspdf-script");

if (html2canvasScript) document.body.removeChild(html2canvasScript);

if (jspdfScript) document.body.removeChild(jspdfScript);

};

}, []);

const handleExportPdf = useCallback(async () => {

if (!pdfLibsLoaded) {

showFeedback("As bibliotecas de PDF ainda estão carregando. Por favor, aguarde e tente novamente.");

return;

}

setIsGeneratingPdf(true);

const reportElement = document.getElementById('report-content');

if (reportElement) {

try {

const canvas = await window.html2canvas(reportElement, {

scale: 2,

useCORS: true,

windowWidth: reportElement.scrollWidth,

windowHeight: reportElement.scrollHeight

});

const imgData = canvas.toDataURL('image/png');

const pdf = new window.jspdf.jsPDF('p', 'mm', 'a4');

const imgWidth = 210;

const pageHeight = 297;

const imgHeight = (canvas.height \* imgWidth) / canvas.width;

let heightLeft = imgHeight;

let position = 0;

pdf.addImage(imgData, 'PNG', 0, position, imgWidth, imgHeight);

heightLeft -= pageHeight;

while (heightLeft >= 0) {

position = heightLeft - imgHeight;

pdf.addPage();

pdf.addImage(imgData, 'PNG', 0, position, imgWidth, imgHeight);

heightLeft -= pageHeight;

}

const date = new Date();

const filename = `Relatorio\_Atividades\_${date.getFullYear()}-${date.getMonth() + 1}-${date.getDate()}.pdf`;

pdf.save(filename);

} catch (error) {

console.error("Erro ao gerar PDF:", error);

showFeedback("Erro ao gerar PDF. Tente novamente.");

} finally {

setIsGeneratingPdf(false);

}

} else {

showFeedback("Conteúdo do relatório não encontrado para exportação.");

setIsGeneratingPdf(false);

}

}, [pdfLibsLoaded, showFeedback]);

return { isGeneratingPdf, pdfLibsLoaded, handleExportPdf };

};

// --- COMPONENTES MODULARES ---

// Componente para o Modal de Edição de Atividade

const EditActivityModal = React.memo(({ activity, onClose, onSave, customStatuses, customRisks, customColumns, currentTheme }) => {

const [editedActivity, setEditedActivity] = useState(activity);

useEffect(() => {

setEditedActivity(activity);

}, [activity]);

const handleChange = useCallback((e) => {

const { name, value, type } = e.target;

let parsedValue = value;

if (type === 'number') {

parsedValue = parseFloat(value) || 0;

}

setEditedActivity(prev => ({

...prev,

[name]: parsedValue

}));

}, []);

const handleSave = useCallback(() => {

onSave(editedActivity);

onClose();

}, [editedActivity, onClose, onSave]);

if (!activity) return null;

const defaultStatuses = ['Concluído', 'Em Andamento', 'Atrasado', 'Não Iniciado'];

const allStatuses = useMemo(() => [...new Set([...defaultStatuses, ...customStatuses.map(s => s.name)])], [customStatuses]);

const defaultRisks = ['Alto', 'Média', 'Baixo'];

const allRisks = useMemo(() => [...new Set([...defaultRisks, ...customRisks.map(r => r.name)])], [customRisks]);

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const labelStyle = { color: currentTheme.secondaryTextColor };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

// Filter out the calculated fields from standard columns for input fields

const standardInputColumns = useMemo(() => columnDefinitions.filter(col =>

col.key !== 'actions' &&

col.key !== 'valorAgregado' && // EV is calculated

col.key !== 'variacaoPrazo' && // SV is calculated

col.key !== 'indiceDesempenhoPrazo' && // SPI is calculated

col.key !== 'variacaoCusto' && // CV is calculated

col.key !== 'indiceDesempenhoCusto' // CPI is calculated

), []);

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-2xl max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Editar Atividade: {editedActivity.atividade}</h2>

<form className="grid grid-cols-1 md:grid-cols-2 gap-4">

{standardInputColumns.map(col => {

if (['dependencias', 'recursosNecessarios', 'observacoes'].includes(col.key)) return null; // Handle these separately

let inputType = col.type;

if (col.key === 'percentualConclusao') inputType = 'number'; // Ensure number for percentage

return (

<div className="col-span-1" key={col.key}>

<label htmlFor={col.key} className="block text-sm font-medium" style={labelStyle}>{col.label}</label>

{col.key === 'prioridade' ? (

<select name={col.key} id={col.key} value={editedActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

<option value="Alta">Alta</option>

<option value="Média">Média</option>

<option value="Baixa">Baixa</option>

</select>

) : col.key === 'statusPlanejado' || col.key === 'statusReal' ? (

<select name={col.key} id={col.key} value={editedActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

{allStatuses.map(status => (<option key={status} value={status}>{status}</option>))}

</select>

) : col.key === 'riscoAssociado' ? (

<select name={col.key} id={col.key} value={editedActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

{allRisks.map(risk => (<option key={risk} value={risk}>{risk}</option>))}

</select>

) : (

<input type={inputType} name={col.key} id={col.key} value={editedActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

</div>

);

})}

{/\* Campos para colunas personalizadas \*/}

{customColumns.map(col => (

<div className="col-span-1" key={col.id}>

<label htmlFor={col.id} className="block text-sm font-medium" style={labelStyle}>{col.name}</label>

{col.type === 'text' && (

<input type="text" name={col.id} id={col.id} value={editedActivity[col.id] || ''} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'number' && (

<input type="number" name={col.id} id={col.id} value={editedActivity[col.id] || 0} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'date' && (

<input type="date" name={col.id} id={col.id} value={editedActivity[col.id] || ''} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'formula' && (

<input type="text" name={col.id} id={col.id} value={activity[col.id] || ''} readOnly disabled className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2 cursor-not-allowed" style={inputStyle} title="Coluna de fórmula é calculada automaticamente" />

)}

</div>

))}

{/\* Campos especiais \*/}

<div className="col-span-full">

<label htmlFor="observacoes" className="block text-sm font-medium" style={labelStyle}>Observações</label>

<textarea name="observacoes" id="observacoes" value={editedActivity.observacoes} onChange={handleChange} rows="3" className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}></textarea>

</div>

<div className="col-span-full">

<label htmlFor="dependencias" className="block text-sm font-medium" style={labelStyle}>Dependências (separadas por vírgula)</label>

<input type="text" name="dependencias" id="dependencias" value={editedActivity.dependencias.join(', ')} onChange={(e) => setEditedActivity(prev => ({ ...prev, dependencias: e.target.value.split(',').map(dep => dep.trim()) }))} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

</div>

<div className="col-span-full">

<label htmlFor="recursosNecessarios" className="block text-sm font-medium" style={labelStyle}>Recursos Necessários (separadas por vírgula)</label>

<input type="text" name="recursosNecessarios" id="recursosNecessarios" value={editedActivity.recursosNecessarios.join(', ')} onChange={(e) => setEditedActivity(prev => ({ ...prev, recursosNecessarios: e.target.value.split(',').map(res => res.trim()) }))} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

</div>

</form>

<div className="mt-6 flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 bg-white text-base font-medium text-gray-700 hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500 sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={handleSave} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary}>Salvar Alterações</button>

</div>

</div>

</div>

);

});

// Componente para o Modal de Adição de Atividade

const AddActivityModal = React.memo(({ onClose, onSave, customStatuses, customRisks, customColumns, currentTheme }) => {

const [newActivity, setNewActivity] = useState(() => {

const initialActivity = {

atividade: '', disciplina: '', dataInicialPlanejada: '', dataFinalPlanejada: '', dataInicialReal: '', dataFinalReal: '',

dataInicialBaseline: '', dataFinalBaseline: '', // New baseline fields

valorPlanejado: 0, valorReal: 0, statusPlanejado: 'Não Iniciado', statusReal: 'Não Iniciado', responsavel: '',

prioridade: 'Média', observacoes: '', dependencias: [], custoReal: 0, recursosNecessarios: [], percentualConclusao: 0,

riscoAssociado: 'Baixo', linkDocumento: '', dataUltimaAtualizacao: new Date().toISOString().split('T')[0],

orcamentoPlanejado: 0, // Initialize new field

};

// Initialize custom columns

customColumns.forEach(col => {

if (col.type === 'number') initialActivity[col.id] = 0;

else if (col.type === 'date') initialActivity[col.id] = '';

else if (col.type === 'text') initialActivity[col.id] = '';

// Formula columns are not initialized here as they are calculated

});

return initialActivity;

});

const handleChange = useCallback((e) => {

const { name, value, type } = e.target;

let parsedValue = value;

if (type === 'number') {

parsedValue = parseFloat(value) || 0;

}

setNewActivity(prev => ({ ...prev, [name]: parsedValue }));

}, []);

const handleSave = useCallback(() => {

onSave({ ...newActivity, id: Date.now() + Math.random() }); // Gera um ID único mais robusto

onClose();

}, [newActivity, onClose, onSave]);

const defaultStatuses = ['Concluído', 'Em Andamento', 'Atrasado', 'Não Iniciado'];

const allStatuses = useMemo(() => [...new Set([...defaultStatuses, ...customStatuses.map(s => s.name)])], [customStatuses]);

const defaultRisks = ['Alto', 'Média', 'Baixo'];

const allRisks = useMemo(() => [...new Set([...defaultRisks, ...customRisks.map(r => r.name)])], [customRisks]);

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const labelStyle = { color: currentTheme.secondaryTextColor };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

// Filter out the calculated fields from standard columns for input fields

const standardInputColumns = useMemo(() => columnDefinitions.filter(col =>

col.key !== 'actions' &&

col.key !== 'valorAgregado' && // EV is calculated

col.key !== 'variacaoPrazo' && // SV is calculated

col.key !== 'indiceDesempenhoPrazo' && // SPI is calculated

col.key !== 'variacaoCusto' && // CV is calculated

col.key !== 'indiceDesempenhoCusto' // CPI is calculated

), []);

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-2xl max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Adicionar Nova Atividade</h2>

<form className="grid grid-cols-1 md:grid-cols-2 gap-4">

{standardInputColumns.map(col => {

if (['dependencias', 'recursosNecessarios', 'observacoes'].includes(col.key)) return null; // Handle these separately

let inputType = col.type;

if (col.key === 'percentualConclusao' || col.key === 'orcamentoPlanejado') inputType = 'number';

return (

<div className="col-span-1" key={col.key}>

<label htmlFor={`new-${col.key}`} className="block text-sm font-medium" style={labelStyle}>{col.label}</label>

{col.key === 'prioridade' ? (

<select name={col.key} id={`new-${col.key}`} value={newActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

<option value="Alta">Alta</option>

<option value="Média">Média</option>

<option value="Baixa">Baixa</option>

</select>

) : col.key === 'statusPlanejado' || col.key === 'statusReal' ? (

<select name={col.key} id={`new-${col.key}`} value={newActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

{allStatuses.map(status => (<option key={status} value={status}>{status}</option>))}

</select>

) : col.key === 'riscoAssociado' ? (

<select name={col.key} id={`new-${col.key}`} value={newActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}>

{allRisks.map(risk => (<option key={risk} value={risk}>{risk}</option>))}

</select>

) : (

<input type={inputType} name={col.key} id={`new-${col.key}`} value={newActivity[col.key]} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

</div>

);

})}

{/\* Campos para colunas personalizadas \*/}

{customColumns.map(col => (

<div className="col-span-1" key={col.id}>

<label htmlFor={`new-${col.id}`} className="block text-sm font-medium" style={labelStyle}>{col.name}</label>

{col.type === 'text' && (

<input type="text" name={col.id} id={`new-${col.id}`} value={newActivity[col.id] || ''} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'number' && (

<input type="number" name={col.id} id={`new-${col.id}`} value={newActivity[col.id] || 0} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'date' && (

<input type="date" name={col.id} id={`new-${col.id}`} value={newActivity[col.id] || ''} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

)}

{col.type === 'formula' && (

<input type="text" name={col.id} id={`new-${col.id}`} value={newActivity[col.id] || ''} readOnly disabled className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2 cursor-not-allowed" style={inputStyle} title="Coluna de fórmula é calculada automaticamente" />

)}

</div>

))}

{/\* Campos especiais \*/}

<div className="col-span-full">

<label htmlFor="new-observacoes" className="block text-sm font-medium" style={labelStyle}>Observações</label>

<textarea name="observacoes" id="new-observacoes" value={newActivity.observacoes} onChange={handleChange} rows="3" className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle}></textarea>

</div>

<div className="col-span-full">

<label htmlFor="new-dependencias" className="block text-sm font-medium" style={labelStyle}>Dependências (separadas por vírgula)</label>

<input type="text" name="dependencias" id="new-dependencias" value={newActivity.dependencias.join(', ')} onChange={(e) => setNewActivity(prev => ({ ...prev, dependencias: e.target.value.split(',').map(dep => dep.trim()) }))} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

</div>

<div className="col-span-full">

<label htmlFor="new-recursosNecessarios" className="block text-sm font-medium" style={labelStyle}>Recursos Necessários (separadas por vírgula)</label>

<input type="text" name="recursosNecessarios" id="new-recursosNecessarios" value={newActivity.recursosNecessarios.join(', ')} onChange={(e) => setNewActivity(prev => ({ ...prev, recursosNecessarios: e.target.value.split(',').map(res => res.trim()) }))} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} />

</div>

</form>

<div className="mt-6 flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 text-base font-medium sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={handleSave} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary}>Adicionar Atividade</button>

</div>

</div>

</div>

);

});

// Componente para o Heatmap

const Heatmap = React.memo(({ data, rowLabels, colLabels, maxCount, currentTheme }) => {

const getColor = useCallback((count) => {

if (maxCount === 0) return currentTheme.tableHeaderBg;

const intensity = count / maxCount;

if (intensity === 0) return currentTheme.tableHeaderBg;

const r = parseInt(currentTheme.primaryColor.substring(1, 3), 16);

const g = parseInt(currentTheme.primaryColor.substring(3, 5), 16);

const b = parseInt(currentTheme.primaryColor.substring(5, 7), 16);

const r\_bg = parseInt(currentTheme.tableHeaderBg.substring(1, 3), 16);

const g\_bg = parseInt(currentTheme.tableHeaderBg.substring(3, 5), 16);

const b\_bg = parseInt(currentTheme.tableHeaderBg.substring(5, 7), 16);

const interpolatedR = Math.round(r\_bg + (r - r\_bg) \* intensity);

const interpolatedG = Math.round(g\_bg + (g - g\_bg) \* intensity);

const interpolatedB = Math.round(b\_bg + (b - b\_bg) \* intensity);

return `rgb(${interpolatedR}, ${interpolatedG}, ${interpolatedB})`;

}, [maxCount, currentTheme]);

return (

<div className="overflow-x-auto">

<div className="inline-flex flex-col min-w-full">

<div className="flex flex-row">

<div className="w-32 flex-shrink-0" style={{ backgroundColor: currentTheme.tableHeaderBg }}></div>

{colLabels.map(label => (

<div key={label} className="flex-1 p-2 text-center text-xs font-semibold border rounded-t-md" style={{ backgroundColor: currentTheme.tableHeaderBg, color: currentTheme.textColor, borderColor: currentTheme.tableBorder }}>

{label}

</div>

))}

</div>

{rowLabels.map(rowLabel => (

<div key={rowLabel} className="flex flex-row">

<div className="w-32 p-2 text-right text-xs font-semibold border rounded-l-md flex-shrink-0" style={{ backgroundColor: currentTheme.tableHeaderBg, color: currentTheme.textColor, borderColor: currentTheme.tableBorder }}>

{rowLabel}

</div>

{colLabels.map(colLabel => (

<div

key={`${rowLabel}-${colLabel}`}

className={`flex-1 p-2 text-center text-sm font-medium border rounded-sm`}

style={{ backgroundColor: getColor(data[rowLabel]?.[colLabel] || 0), color: getLuminance(getColor(data[rowLabel]?.[colLabel] || 0)) > 0.5 ? '#1F2937' : '#FFFFFF', borderColor: currentTheme.tableBorder }}

>

{data[rowLabel]?.[colLabel] || 0}

</div>

))}

</div>

))}

</div>

</div>

);

});

// Componente para o Modal de Seleção/Criação de Dashboard

const DashboardSelectorModal = React.memo(({ currentTheme, onSelectDashboard, onClose }) => {

const { db, auth, userId, isAuthReady, setActiveDashboardId } = useDashboardData();

const appId = typeof \_\_app\_id !== 'undefined' ? \_\_app\_id : 'default-app-id';

const [dashboards, setDashboards] = useState([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState('');

const [newDashboardName, setNewDashboardName] = useState('');

const [newDashboardPassword, setNewDashboardPassword] = useState('');

const [selectedDashboardId, setSelectedDashboardId] = useState(null);

const [passwordInput, setPasswordInput] = useState('');

const [showPasswordPrompt, setShowPasswordPrompt] = useState(false);

const [passwordError, setPasswordError] = useState('');

useEffect(() => {

if (db && isAuthReady) {

const dashboardsColRef = collection(db, `artifacts/${appId}/public/data/dashboards`);

const unsubscribe = onSnapshot(dashboardsColRef, (snapshot) => {

const fetchedDashboards = snapshot.docs.map(doc => ({

id: doc.id,

...doc.data()

}));

setDashboards(fetchedDashboards);

setLoading(false);

}, (err) => {

console.error("Erro ao buscar dashboards:", err);

setError("Falha ao carregar dashboards.");

setLoading(false);

});

return () => unsubscribe();

}

}, [db, isAuthReady, appId]);

const handleCreateDashboard = useCallback(async () => {

setError('');

setPasswordError('');

if (!newDashboardName.trim() || !newDashboardPassword.trim()) {

setError('Nome e senha do dashboard não podem ser vazios.');

return;

}

if (!db || !userId) {

setError('Firebase não inicializado ou usuário não autenticado.');

return;

}

const q = query(collection(db, `artifacts/${appId}/public/data/dashboards`), where("name", "==", newDashboardName.trim()));

const querySnapshot = await getDocs(q);

if (!querySnapshot.empty) {

setError('Já existe um dashboard com este nome. Por favor, escolha outro.');

return;

}

try {

setLoading(true);

const dashboardRef = doc(collection(db, `artifacts/${appId}/public/data/dashboards`));

await setDoc(dashboardRef, {

name: newDashboardName.trim(),

password: newDashboardPassword.trim(),

activities: [], customStatuses: [], customRisks: [], customColumns: [], customCharts: [], // Inicializa novos campos

createdAt: new Date().toISOString(), createdBy: userId,

});

onSelectDashboard(dashboardRef.id);

setNewDashboardName('');

setNewDashboardPassword('');

onClose();

} catch (err) {

console.error("Erro ao criar dashboard:", err);

setError("Falha ao criar o dashboard. Tente novamente.");

} finally {

setLoading(false);

}

}, [newDashboardName, newDashboardPassword, db, userId, appId, onSelectDashboard, onClose]);

const handleOpenDashboard = useCallback((dashboard) => {

setSelectedDashboardId(dashboard.id);

setShowPasswordPrompt(true);

}, []);

const handlePasswordSubmit = useCallback(() => {

setPasswordError('');

const selectedDash = dashboards.find(d => d.id === selectedDashboardId);

if (selectedDash && selectedDash.password === passwordInput) {

onSelectDashboard(selectedDashboardId);

onClose();

} else {

setPasswordError('Senha incorreta.');

}

}, [dashboards, selectedDashboardId, passwordInput, onSelectDashboard, onClose]);

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

if (loading || !isAuthReady) {

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-md" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<p className="text-center">Carregando dashboards...</p>

</div>

</div>

);

}

if (error) {

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-md" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<p className="text-center text-red-500">{error}</p>

<div className="mt-4 flex justify-end">

<button onClick={onClose} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={buttonStylePrimary}>Fechar</button>

</div>

</div>

</div>

);

}

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-2xl max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Selecionar ou Criar Dashboard</h2>

<div className="mb-6">

<h3 className="text-lg font-semibold mb-2" style={{ color: currentTheme.textColor }}>Dashboards Existentes:</h3>

{dashboards.length === 0 ? (

<p className="text-sm italic" style={{ color: currentTheme.secondaryTextColor }}>Nenhum dashboard encontrado. Crie um novo!</p>

) : (

<ul className="space-y-2">

{dashboards.map(dashboard => (

<li key={dashboard.id} className="flex justify-between items-center p-3 rounded-md border" style={{ borderColor: currentTheme.tableBorder, backgroundColor: currentTheme.background }}>

<span className="font-medium" style={{ color: currentTheme.textColor }}>{dashboard.name}</span>

<button onClick={() => handleOpenDashboard(dashboard)} className="px-3 py-1 rounded-md shadow-sm transition-colors duration-200" style={buttonStylePrimary}>Abrir</button>

</li>

))}

</ul>

)}

</div>

{showPasswordPrompt && (

<div className="mb-6 p-4 rounded-lg shadow-inner" style={{ backgroundColor: currentTheme.background }}>

<h3 className="text-lg font-semibold mb-2" style={{ color: currentTheme.textColor }}>Digite a senha para "{dashboards.find(d => d.id === selectedDashboardId)?.name}":</h3>

<input type="password" className="mt-1 block w-full rounded-md border-gray-300 shadow-sm focus:border-blue-500 focus:ring-blue-500 sm:text-sm p-2" style={inputStyle} value={passwordInput} onChange={(e) => setPasswordInput(e.target.value)} onKeyPress={(e) => { if (e.key === 'Enter') handlePasswordSubmit(); }} />

{passwordError && <p className="text-red-500 text-sm mt-1">{passwordError}</p>}

<div className="mt-4 flex justify-end gap-3">

<button onClick={() => setShowPasswordPrompt(false)} className="px-4 py-2 rounded-md border border-gray-300 shadow-sm transition-colors duration-200" style={buttonStyleSecondary}>Cancelar</button>

<button onClick={handlePasswordSubmit} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={buttonStylePrimary}>Confirmar</button>

</div>

</div>

)}

<div>

<h3 className="text-lg font-semibold mb-2" style={{ color: currentTheme.textColor }}>Criar Novo Dashboard:</h3>

<div className="flex flex-col gap-3">

<input type="text" placeholder="Nome do Novo Dashboard" className="p-2 border rounded-md" style={inputStyle} value={newDashboardName} onChange={(e) => setNewDashboardName(e.target.value)} />

<input type="password" placeholder="Senha do Novo Dashboard" className="p-2 border rounded-md" style={inputStyle} value={newDashboardPassword} onChange={(e) => setNewDashboardPassword(e.target.value)} />

{error && <p className="text-red-500 text-sm mt-1">{error}</p>}

<button onClick={handleCreateDashboard} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={{ backgroundColor: currentTheme.secondaryColor, color: currentTheme.textColor }}>Criar Dashboard</button>

</div>

</div>

<div className="mt-6 flex justify-end">

<button onClick={onClose} className="px-4 py-2 rounded-md border border-gray-300 shadow-sm transition-colors duration-200" style={buttonStyleSecondary}>Fechar</button>

</div>

</div>

</div>

);

});

// Componente para a pré-visualização do CSV

const CsvPreviewModal = React.memo(({ headers, data, onClose, onConfirm, currentTheme }) => {

const tableHeaderStyle = { backgroundColor: currentTheme.tableHeaderBg };

const tableBodyStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-4xl max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Pré-visualização do CSV</h2>

{data.length === 0 ? (

<p className="text-center text-sm italic" style={{ color: currentTheme.secondaryTextColor }}>Nenhum dado válido para pré-visualizar.</p>

) : (

<div className="overflow-x-auto rounded-lg border" style={{ borderColor: currentTheme.tableBorder }}>

<table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}>

<thead style={tableHeaderStyle}>

<tr>

{headers.map((header, index) => (

<th key={index} className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryTextColor }}>

{header}

</th>

))}

</tr>

</thead>

<tbody style={tableBodyStyle}>

{data.slice(0, 5).map((row, rowIndex) => (

<tr key={rowIndex}>

{headers.map((header, colIndex) => (

<td key={colIndex} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.textColor }}>

{row[header]}

</td>

))}

</tr>

))}

</tbody>

</table>

{data.length > 5 && (

<p className="text-center text-sm italic mt-2" style={{ color: currentTheme.secondaryTextColor }}>

... e mais {data.length - 5} linhas.

</p>

)}

</div>

)}

<p className="text-sm mt-4" style={{ color: currentTheme.secondaryTextColor }}>

Confirme a importação das atividades acima. Campos vazios ou inválidos podem ser preenchidos com valores padrão.

</p>

<div className="mt-6 flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 text-base font-medium sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={onConfirm} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary} disabled={data.length === 0}>Confirmar Importação</button>

</div>

</div>

</div>

);

});

// Componente para os Cards de Visão Geral

const OverviewCards = React.memo(({ metrics, currentTheme }) => {

const {

plannedSummary, realSummary, overallCompletionPercentage, completedActivities, totalActivities,

totalPlannedCost, totalRealCost, costDeviation, totalDiasAtraso, totalDiasRemanescentes,

totalOrcamentoPlanejado, totalValorAgregado,

overallVariacaoPrazo, overallIndiceDesempenhoPrazo, overallVariacaoCusto, overallIndiceDesempenhoCusto

} = metrics;

const cardStyle = (borderColor) => ({ backgroundColor: currentTheme.cardBackground, borderColor });

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

// Helper for KPI color based on value

const getKpiColor = (value, type) => {

if (type === 'index') { // SPI, CPI

return value >= 1 ? currentTheme.secondaryColor : currentTheme.dangerColor;

} else if (type === 'variance') { // SV, CV

return value >= 0 ? currentTheme.secondaryColor : currentTheme.dangerColor;

}

return currentTheme.textColor;

};

return (

<section className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8 no-print">

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(currentTheme.primaryColor)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Status Planejado</h2>

{Object.entries(plannedSummary).map(([status, count]) => (

<p key={status} style={secondaryTextColor}>

<span className="font-medium">{status}:</span> {count} atividades

</p>

))}

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(currentTheme.secondaryColor)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Status Real</h2>

{Object.entries(realSummary).map(([status, count]) => (

<p key={status} style={secondaryTextColor}>

<span className="font-medium">{status}:</span> {count} atividades

</p>

))}

</div>

<div className="p-6 rounded-lg shadow-md border-l-4 col-span-1 md:col-span-2 lg:col-span-1" style={cardStyle(currentTheme.accentColor)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Conclusão Geral</h2>

<div className="flex items-center">

<div className="w-full rounded-full h-4" style={{ backgroundColor: currentTheme.tableBorder }}>

<div className="h-4 rounded-full transition-all duration-500 ease-out" style={{ width: `${overallCompletionPercentage}%`, backgroundColor: currentTheme.accentColor }}></div>

</div>

<span className="ml-4 text-lg font-bold" style={{ color: currentTheme.accentColor }}>

{overallCompletionPercentage.toFixed(1)}%

</span>

</div>

<p className="mt-2" style={secondaryTextColor}>

{completedActivities} de {totalActivities} atividades concluídas.

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(currentTheme.dangerColor)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Desvio de Custos</h2>

<p style={secondaryTextColor}>

<span className="font-medium">Total Planejado:</span> {formatCurrency(totalPlannedCost)}

</p>

<p style={secondaryTextColor}>

<span className="font-medium">Total Real:</span> {formatCurrency(totalRealCost)}

</p>

<p className="font-bold mt-2" style={textColor}>

Desvio: {formatCurrency(costDeviation)}

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(currentTheme.warningColor)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Total de Dias de Atraso</h2>

<p className="text-2xl font-bold" style={{ color: currentTheme.warningColor }}>{totalDiasAtraso} dias</p>

<p className="mt-2" style={secondaryTextColor}>

Soma dos dias de atraso de todas as atividades.

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(currentTheme.chartSecondary)}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>Total de Dias Remanescentes</h2>

<p className="text-2xl font-bold" style={{ color: currentTheme.chartSecondary }}>{totalDiasRemanescentes} dias</p>

<p className="mt-2" style={secondaryTextColor}>

Soma dos dias restantes para atividades não concluídas.

</p>

</div>

{/\* New EVM KPIs \*/}

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(getKpiColor(overallIndiceDesempenhoPrazo, 'index'))}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>SPI (Índice de Desempenho de Prazo)</h2>

<p className="text-2xl font-bold" style={{ color: getKpiColor(overallIndiceDesempenhoPrazo, 'index') }}>

{overallIndiceDesempenhoPrazo.toFixed(2)}

</p>

<p className="mt-2" style={secondaryTextColor}>

{overallIndiceDesempenhoPrazo >= 1 ? 'Adiantado' : 'Atrasado'} em relação ao prazo.

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(getKpiColor(overallVariacaoPrazo, 'variance'))}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>SV (Variação de Prazo)</h2>

<p className="text-2xl font-bold" style={{ color: getKpiColor(overallVariacaoPrazo, 'variance') }}>

{formatCurrency(overallVariacaoPrazo)}

</p>

<p className="mt-2" style={secondaryTextColor}>

{overallVariacaoPrazo >= 0 ? 'À frente' : 'Atrás'} do cronograma em valor.

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(getKpiColor(overallIndiceDesempenhoCusto, 'index'))}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>CPI (Índice de Desempenho de Custo)</h2>

<p className="text-2xl font-bold" style={{ color: getKpiColor(overallIndiceDesempenhoCusto, 'index') }}>

{overallIndiceDesempenhoCusto.toFixed(2)}

</p>

<p className="mt-2" style={secondaryTextColor}>

{overallIndiceDesempenhoCusto >= 1 ? 'Abaixo do Orçamento' : 'Acima do Orçamento'}.

</p>

</div>

<div className="p-6 rounded-lg shadow-md border-l-4" style={cardStyle(getKpiColor(overallVariacaoCusto, 'variance'))}>

<h2 className="text-xl font-semibold mb-4" style={textColor}>CV (Variação de Custo)</h2>

<p className="text-2xl font-bold" style={{ color: getKpiColor(overallVariacaoCusto, 'variance') }}>

{formatCurrency(overallVariacaoCusto)}

</p>

<p className="mt-2" style={secondaryTextColor}>

{overallVariacaoCusto >= 0 ? 'Abaixo' : 'Acima'} do orçamento em valor.

</p>

</div>

</section>

);

});

// Componente para a Seção de Filtros de Data

const DateFilterSection = React.memo(({ startDate, setStartDate, endDate, setEndDate, selectedPeriod, setSelectedPeriod, currentTheme }) => {

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const labelStyle = { color: currentTheme.secondaryTextColor };

// Efeito para ajustar as datas com base no período selecionado

useEffect(() => {

const today = new Date();

let newStartDate = '';

let newEndDate = '';

const getFormattedDate = (date) => date.toISOString().split('T')[0];

switch (selectedPeriod) {

case 'Semana':

const firstDayOfWeek = new Date(today.setDate(today.getDate() - today.getDay()));

newStartDate = getFormattedDate(firstDayOfWeek);

newEndDate = getFormattedDate(new Date(firstDayOfWeek.setDate(firstDayOfWeek.getDate() + 6)));

break;

case 'Mês':

newStartDate = getFormattedDate(new Date(today.getFullYear(), today.getMonth(), 1));

newEndDate = getFormattedDate(new Date(today.getFullYear(), today.getMonth() + 1, 0));

break;

case 'Trimestre':

const currentMonth = today.getMonth();

const currentQuarter = Math.floor(currentMonth / 3);

newStartDate = getFormattedDate(new Date(today.getFullYear(), currentQuarter \* 3, 1));

newEndDate = getFormattedDate(new Date(today.getFullYear(), currentQuarter \* 3 + 3, 0));

break;

case 'Semestre':

const currentSemester = today.getMonth() < 6 ? 0 : 6;

newStartDate = getFormattedDate(new Date(today.getFullYear(), currentSemester, 1));

newEndDate = getFormattedDate(new Date(today.getFullYear(), currentSemester + 6, 0));

break;

case 'Ano':

newStartDate = getFormattedDate(new Date(today.getFullYear(), 0, 1));

newEndDate = getFormattedDate(new Date(today.getFullYear(), 11, 31));

break;

case 'Todos':

default:

newStartDate = '';

newEndDate = '';

break;

}

setStartDate(newStartDate);

setEndDate(newEndDate);

}, [selectedPeriod, setStartDate, setEndDate]);

return (

<section className="p-6 rounded-lg shadow-md mb-8 no-print" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={{ color: currentTheme.textColor }}>Filtrar por Período</h2>

<div className="flex flex-wrap gap-4 items-center">

<div className="flex flex-col flex-grow sm:flex-grow-0">

<label htmlFor="period-select" className="text-sm font-medium" style={labelStyle}>Período Rápido:</label>

<select id="period-select" className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full md:w-auto" style={inputStyle} value={selectedPeriod} onChange={(e) => setSelectedPeriod(e.target.value)}>

<option value="Todos">Todos os Períodos</option>

<option value="Semana">Esta Semana</option>

<option value="Mês">Este Mês</option>

<option value="Trimestre">Este Trimestre</option>

<option value="Semestre">Este Semestre</option>

<option value="Ano">Este Ano</option>

</select>

</div>

<div className="flex flex-col flex-grow sm:flex-grow-0">

<label htmlFor="start-date" className="text-sm font-medium" style={labelStyle}>Data Inicial:</label>

<input type="date" id="start-date" className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full md:w-auto" style={inputStyle} value={startDate} onChange={(e) => { setStartDate(e.target.value); setSelectedPeriod('Customizado'); }} />

</div>

<div className="flex flex-col flex-grow sm:flex-grow-0">

<label htmlFor="end-date" className="text-sm font-medium" style={labelStyle}>Data Final:</label>

<input type="date" id="end-date" className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full md:w-auto" style={inputStyle} value={endDate} onChange={(e) => { setEndDate(e.target.value); setSelectedPeriod('Customizado'); }} />

</div>

</div>

</section>

);

});

// Componente para exibir um gráfico personalizado

const CustomChartDisplay = React.memo(({ chart, data, currentTheme, getStatusBgColor }) => {

const chartTitleStyle = { color: currentTheme.textColor };

const chartAxisStyle = { stroke: currentTheme.secondaryTextColor };

const tooltipStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, border: `1px solid ${currentTheme.tableBorder}` };

const legendStyle = { color: currentTheme.textColor };

const gridStyle = { strokeDasharray: "3 3", stroke: currentTheme.tableBorder };

if (!chart || !chart.xAxis || !chart.yAxis) {

return <div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.dangerColor }}>Configuração de gráfico inválida.</div>;

}

const yAxisKeys = Array.isArray(chart.yAxis) ? chart.yAxis : [chart.yAxis];

const chartData = useMemo(() => {

return data.map(item => {

const newItem = { [chart.xAxis]: item[chart.xAxis] };

yAxisKeys.forEach(key => {

// Ensure that values are numbers for charting, convert 'Erro' to 0

newItem[key] = typeof item[key] === 'number' ? item[key] : (item[key] === 'Erro' ? 0 : parseFloat(item[key]) || 0);

});

return newItem;

});

}, [data, chart.xAxis, yAxisKeys]);

const getFillColor = (index) => {

// Attempt to use status color if yAxisKey is a status column

if (chart.xAxis === 'statusReal' || chart.xAxis === 'statusPlanejado') {

const statusName = chartData[index]?.[chart.xAxis];

return getStatusBgColor(statusName);

}

// Fallback to a color from the theme or a default palette

const themeColors = [currentTheme.chartPrimary, currentTheme.chartSecondary, currentTheme.chartAccent, currentTheme.chartPriority, currentTheme.chartRisk];

return themeColors[index % themeColors.length] || PIE\_COLORS[index % PIE\_COLORS.length];

};

return (

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>{chart.name}</h2>

<ResponsiveContainer width="100%" height={300}>

{chart.type === 'BarChart' ? (

<BarChart data={chartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey={chart.xAxis} style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

{yAxisKeys.map((key, index) => (

<Bar key={key} dataKey={key} fill={getFillColor(index)} name={key} stackId={chart.stackBars ? "a" : undefined} radius={[10, 10, 0, 0]} />

))}

</BarChart>

) : chart.type === 'PieChart' ? (

<PieChart>

<Pie

data={chartData}

dataKey={yAxisKeys[0]} // Pie chart usually has one value key

nameKey={chart.xAxis} // Name key for labels

cx="50%"

cy="50%"

outerRadius={100}

fill={currentTheme.chartPrimary}

label={({ name, percent }) => `${name}: ${(percent \* 100).toFixed(0)}%`}

labelLine={false}

>

{chartData.map((entry, index) => (

<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />

))}

</Pie>

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

</PieChart>

) : chart.type === 'LineChart' ? (

<LineChart data={chartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey={chart.xAxis} style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

{yAxisKeys.map((key, index) => (

<Line key={key} type="monotone" dataKey={key} stroke={getFillColor(index)} name={key} />

))}

</LineChart>

) : (

<p style={{ color: currentTheme.dangerColor }}>Tipo de gráfico não suportado: {chart.type}</p>

)}

</ResponsiveContainer>

</div>

);

});

// Componente para a Seção de Gráficos

const ChartsDisplaySection = React.memo(({ metrics, customCharts, currentTheme, getStatusBgColor }) => {

const {

statusChartData, deadlineDeviationData, costChartData, completionPieData,

priorityChartData, riskChartData, heatmapData, allDisciplinesForHeatmap, allResponsaveisForHeatmap, maxActivityCount,

disciplineStatusData, responsibleStatusData, allPossibleStatuses, activitiesWithCalculatedMetrics

} = metrics;

const chartTitleStyle = { color: currentTheme.textColor };

const chartAxisStyle = { stroke: currentTheme.secondaryTextColor };

const tooltipStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, border: `1px solid ${currentTheme.tableBorder}` };

const legendStyle = { color: currentTheme.textColor };

const gridStyle = { strokeDasharray: "3 3", stroke: currentTheme.tableBorder };

return (

<section className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8 no-print">

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Comparativo de Status (Planejado vs. Real)</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={statusChartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="status" style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

<Bar dataKey="Planejado" fill={currentTheme.chartPrimary} name="Planejado" radius={[10, 10, 0, 0]} />

<Bar dataKey="Real" fill={currentTheme.chartSecondary} name="Real" radius={[10, 10, 0, 0]} />

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Desvio de Prazos (Dias)</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={deadlineDeviationData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="atividade" hide={true} />

<YAxis style={chartAxisStyle} />

<Tooltip formatter={(value) => `${value} dias`} contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

<Bar dataKey="desvioDias" fill={currentTheme.chartAccent} name="Desvio em Dias" radius={[10, 10, 0, 0]} />

</BarChart>

</ResponsiveContainer>

<p className="text-sm mt-2 text-center" style={{ color: currentTheme.secondaryTextColor }}>

(Valores positivos indicam atraso, negativos indicam adiantamento)

</p>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Comparativo de Custos por Disciplina</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={costChartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="disciplina" style={chartAxisStyle} />

<YAxis tickFormatter={(value) => formatCurrency(value)} style={chartAxisStyle} />

<Tooltip formatter={(value) => formatCurrency(value)} contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

<Bar dataKey="Planejado" fill={currentTheme.chartPrimary} name="Planejado" radius={[10, 10, 0, 0]} />

<Bar dataKey="Real" fill={currentTheme.chartSecondary} name="Real" radius={[10, 10, 0, 0]} />

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Conclusão por Disciplina (%)</h2>

<ResponsiveContainer width="100%" height={300}>

<PieChart>

<Pie

data={completionPieData}

cx="50%"

cy="50%"

labelLine={false}

outerRadius={100}

fill={currentTheme.chartPrimary}

dataKey="value"

label={({ name, percent }) => `${name}: ${(percent \* 100).toFixed(0)}%`}

>

{completionPieData.map((entry, index) => (

<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />

))}

</Pie>

<Tooltip formatter={(value) => `${value.toFixed(1)}%`} contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

</PieChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Atividades por Prioridade</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={priorityChartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="name" style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

<Bar dataKey="count" fill={currentTheme.chartPriority} name="Número de Atividades" radius={[10, 10, 0, 0]} />

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Atividades por Risco</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={riskChartData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="name" style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

<Bar dataKey="count" fill={currentTheme.chartRisk} name="Número de Atividades" radius={[10, 10, 0, 0]} />

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md lg:col-span-2" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Heatmap: Atividades por Responsável e Disciplina</h2>

<Heatmap

data={heatmapData}

rowLabels={allDisciplinesForHeatmap}

colLabels={allResponsaveisForHeatmap}

maxCount={maxActivityCount}

currentTheme={currentTheme}

/>

<p className="text-sm mt-2 text-center" style={{ color: currentTheme.secondaryTextColor }}>

(A intensidade da cor indica o número de atividades)

</p>

</div>

{/\* Novos Gráficos Cruzados \*/}

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Atividades por Disciplina e Status</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={disciplineStatusData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="disciplina" style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

{allPossibleStatuses.map(status => (

<Bar key={status} dataKey={status} stackId="a" fill={getStatusBgColor(status)} name={status} />

))}

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-md" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={chartTitleStyle}>Atividades por Responsável e Status</h2>

<ResponsiveContainer width="100%" height={300}>

<BarChart data={responsibleStatusData} margin={{ top: 20, right: 30, left: 20, bottom: 5 }}>

<CartesianGrid style={gridStyle} />

<XAxis dataKey="responsavel" style={chartAxisStyle} />

<YAxis style={chartAxisStyle} />

<Tooltip contentStyle={tooltipStyle} />

<Legend wrapperStyle={legendStyle} />

{allPossibleStatuses.map(status => (

<Bar key={status} dataKey={status} stackId="a" fill={getStatusBgColor(status)} name={status} />

))}

</BarChart>

</ResponsiveContainer>

</div>

{/\* Gráficos personalizados \*/}

{customCharts.map(chart => (

<CustomChartDisplay

key={chart.id}

chart={chart}

data={activitiesWithCalculatedMetrics}

currentTheme={currentTheme}

getStatusBgColor={getStatusBgColor}

/>

))}

</section>

);

});

// Componente para a Tabela de Atividades

const ActivityTableSection = React.memo(({

activitiesWithCalculatedMetrics, uniqueResponsaveis, customStatuses, customRisks, customColumns,

currentTheme, searchTerm, setSearchTerm, filterStatus, setFilterStatus,

filterResponsavel, setFilterResponsavel, handleEditClick, handleDeleteClick, visibleColumns,

getStatusBgColor, getStatusTextColor, getRiskBgColor, getRiskTextColor

}) => {

const tableHeaderStyle = { backgroundColor: currentTheme.tableHeaderBg };

const tableBodyStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const buttonStyleEdit = { color: currentTheme.primaryColor, borderColor: currentTheme.primaryColor, backgroundColor: currentTheme.cardBackground };

const buttonStyleDelete = { color: currentTheme.dangerColor, borderColor: currentTheme.dangerColor, backgroundColor: currentTheme.cardBackground };

const allDisplayColumns = useMemo(() => {

const standardCols = columnDefinitions.filter(col => col.key !== 'actions');

const customCols = customColumns.map(col => ({ key: col.id, label: col.name, type: col.type }));

return [...standardCols, ...customCols, { key: 'actions', label: 'Ações', type: 'action' }];

}, [customColumns]);

return (

<div className="no-print">

{/\* Filtros da Tabela \*/}

<div className="flex flex-wrap gap-4 mb-6 items-center">

<input type="text" placeholder="Buscar atividade, disciplina, etc." className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 flex-grow" style={inputStyle} value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} />

<select className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full md:w-auto" style={inputStyle} value={filterStatus} onChange={(e) => setFilterStatus(e.target.value)}>

<option value="Todos">Filtrar por Status Real</option>

{[...new Set(['Concluído', 'Em Andamento', 'Atrasado', 'Não Iniciado', ...customStatuses.map(s => s.name)])].map(status => (

<option key={status} value={status}>{status}</option>

))}

</select>

<select className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full md:w-auto" style={inputStyle} value={filterResponsavel} onChange={(e) => setFilterResponsavel(e.target.value)}>

<option value="Todos">Filtrar por Responsável</option>

{uniqueResponsaveis.map(responsavel => (

<option key={responsavel} value={responsavel}>{responsavel}</option>

))}

</select>

</div>

{/\* Tabela Responsiva \*/}

<div className="overflow-x-auto rounded-lg border" style={{ borderColor: currentTheme.tableBorder }}>

<table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}>

<thead style={tableHeaderStyle}>

<tr>

{allDisplayColumns.map(col => (

visibleColumns[col.key] && (

<th key={col.key} scope="col" className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryTextColor }}>

{col.label}

</th>

)

))}

</tr>

</thead>

<tbody style={tableBodyStyle}>

{activitiesWithCalculatedMetrics.length > 0 ? (

activitiesWithCalculatedMetrics.map((activity) => (

<tr key={activity.id}>

{allDisplayColumns.map(col => {

if (!visibleColumns[col.key]) return null;

switch (col.key) {

case 'atividade': case 'disciplina': case 'responsavel': case 'prioridade': case 'linkDocumento':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={{ color: currentTheme.textColor }}>{activity[col.key]}</td>;

case 'dataInicialPlanejada': case 'dataFinalPlanejada': case 'dataInicialReal': case 'dataFinalReal': case 'dataUltimaAtualizacao':

case 'dataInicialBaseline': case 'dataFinalBaseline': // Render baseline dates

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryTextColor }}>{formatDate(activity[col.key])}</td>;

case 'valorPlanejado': case 'valorReal': case 'custoReal': case 'orcamentoPlanejado': case 'valorAgregado': case 'variacaoPrazo': case 'variacaoCusto':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryTextColor }}>{formatCurrency(activity[col.key])}</td>;

case 'statusPlanejado':

return (<td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm"><span className="px-2 inline-flex text-xs leading-5 font-semibold rounded-full" style={{ backgroundColor: getStatusBgColor(activity.statusPlanejado), color: getStatusTextColor(activity.statusPlanejada) }}>{activity.statusPlanejado}</span></td>);

case 'statusReal':

return (<td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm"><span className="px-2 inline-flex text-xs leading-5 font-semibold rounded-full" style={{ backgroundColor: getStatusBgColor(activity.statusReal), color: getStatusTextColor(activity.statusReal) }}>{activity.statusReal}</span></td>);

case 'percentualConclusao':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryTextColor }}>{activity[col.key]}%</td>;

case 'riscoAssociado':

return (<td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm"><span className="px-2 inline-flex text-xs leading-5 font-semibold rounded-full" style={{ backgroundColor: getRiskBgColor(activity.riscoAssociado), color: getRiskTextColor(activity.riscoAssociado) }}>{activity.riscoAssociado}</span></td>);

case 'diasAtraso':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: activity.diasAtraso > 0 ? currentTheme.dangerColor : currentTheme.secondaryTextColor }}>{activity.diasAtraso}</td>;

case 'diasRemanescentes':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: activity.diasRemanescentes > 0 ? currentTheme.secondaryColor : currentTheme.secondaryTextColor }}>{activity.diasRemanescentes}</td>;

case 'indiceDesempenhoPrazo':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: activity.indiceDesempenhoPrazo >= 1 ? currentTheme.secondaryColor : currentTheme.dangerColor }}>{activity.indiceDesempenhoPrazo.toFixed(2)}</td>;

case 'indiceDesempenhoCusto':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: activity.indiceDesempenhoCusto >= 1 ? currentTheme.secondaryColor : currentTheme.dangerColor }}>{activity.indiceDesempenhoCusto.toFixed(2)}</td>;

case 'recursosNecessarios':

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryTextColor }}>{Array.isArray(activity[col.key]) ? activity[col.key].join(', ') : activity[col.key]}</td>;

case 'actions':

return (<td key={col.key} className="px-6 py-4 whitespace-nowrap text-right text-sm font-medium">

<button onClick={() => handleEditClick(activity)} className="px-3 py-1 rounded-md border transition-colors duration-200 mr-2" style={buttonStyleEdit}>Editar</button>

<button onClick={() => handleDeleteClick(activity)} className="px-3 py-1 rounded-md border transition-colors duration-200" style={buttonStyleDelete}>Excluir</button>

</td>);

default:

// Handle custom columns

const customColDef = customColumns.find(c => c.id === col.key);

if (customColDef) {

let displayValue = activity[col.key];

if (customColDef.type === 'number' || customColDef.type === 'formula') {

displayValue = typeof displayValue === 'number' ? displayValue.toFixed(2) : displayValue;

} else if (customColDef.type === 'date') {

displayValue = formatDate(displayValue);

}

return <td key={col.key} className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryTextColor }}>{displayValue}</td>;

}

return null;

}

})}

</tr>

))

) : (

<tr>

<td colSpan={allDisplayColumns.length} className="px-6 py-4 text-center text-gray-500" style={{ color: currentTheme.secondaryTextColor }}>

Nenhuma atividade encontrada com os filtros aplicados.

</td>

</tr>

)}

</tbody>

</table>

</div>

</div>

);

});

// Componente para o Modal de Criação de Gráfico

const CreateChartModal = React.memo(({ onClose, onSave, allAvailableColumns, currentTheme }) => {

const [chartName, setChartName] = useState('');

const [chartType, setChartType] = useState('BarChart');

const [xAxis, setXAxis] = useState('');

const [yAxis, setYAxis] = useState([]); // Can be multiple for stacked bars

const [stackBars, setStackBars] = useState(false);

const [error, setError] = useState('');

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const labelStyle = { color: currentTheme.secondaryTextColor };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const availableXAxisColumns = useMemo(() =>

allAvailableColumns.filter(col => col.type === 'text' || col.type === 'date' || col.type === 'number' || col.type === 'formula')

, [allAvailableColumns]);

const availableYAxisColumns = useMemo(() =>

allAvailableColumns.filter(col => col.type === 'number' || col.type === 'formula')

, [allAvailableColumns]);

const handleYAxisChange = useCallback((e) => {

const { options } = e.target;

const selectedValues = [];

for (let i = 0, l = options.length; i < l; i++) {

if (options[i].selected) {

selectedValues.push(options[i].value);

}

}

setYAxis(selectedValues);

}, []);

const handleSaveChart = useCallback(() => {

setError('');

if (!chartName.trim() || !xAxis || yAxis.length === 0) {

setError('Por favor, preencha todos os campos obrigatórios.');

return;

}

onSave({

id: Date.now() + Math.random(),

name: chartName.trim(),

type: chartType,

xAxis: xAxis,

yAxis: yAxis,

stackBars: stackBars,

});

onClose();

}, [chartName, chartType, xAxis, yAxis, stackBars, onClose, onSave]);

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-2xl max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Criar Novo Gráfico</h2>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div className="col-span-full">

<label htmlFor="chart-name" className="block text-sm font-medium" style={labelStyle}>Nome do Gráfico</label>

<input type="text" id="chart-name" value={chartName} onChange={(e) => setChartName(e.target.value)} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle} />

</div>

<div className="col-span-1">

<label htmlFor="chart-type" className="block text-sm font-medium" style={labelStyle}>Tipo de Gráfico</label>

<select id="chart-type" value={chartType} onChange={(e) => setChartType(e.target.value)} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle}>

<option value="BarChart">Gráfico de Barras</option>

<option value="PieChart">Gráfico de Pizza</option>

<option value="LineChart">Gráfico de Linha</option> {/\* Added LineChart option \*/}

</select>

</div>

<div className="col-span-1">

<label htmlFor="x-axis" className="block text-sm font-medium" style={labelStyle}>Eixo X (Categoria)</label>

<select id="x-axis" value={xAxis} onChange={(e) => setXAxis(e.target.value)} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle}>

<option value="">Selecione uma coluna</option>

{availableXAxisColumns.map(col => (

<option key={col.key} value={col.key}>{col.label}</option>

))}

</select>

</div>

<div className="col-span-full">

<label htmlFor="y-axis" className="block text-sm font-medium" style={labelStyle}>Eixo Y (Valores - Selecione um ou mais)</label>

<select multiple id="y-axis" value={yAxis} onChange={handleYAxisChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2 h-32" style={inputStyle}>

{availableYAxisColumns.map(col => (

<option key={col.key} value={col.key}>{col.label}</option>

))}

</select>

</div>

{chartType === 'BarChart' && yAxis.length > 1 && (

<div className="col-span-full">

<label className="inline-flex items-center text-sm" style={labelStyle}>

<input type="checkbox" checked={stackBars} onChange={(e) => setStackBars(e.target.checked)} className="form-checkbox h-4 w-4 rounded" style={{ color: currentTheme.primaryColor }} />

<span className="ml-2">Empilhar Barras</span>

</label>

</div>

)}

</div>

{error && <p className="text-red-500 text-sm mt-4">{error}</p>}

<div className="mt-6 flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 text-base font-medium sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={handleSaveChart} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary}>Salvar Gráfico</button>

</div>

</div>

</div>

);

});

// Componente para o Modal de Confirmação de Exclusão

const DeleteConfirmationModal = React.memo(({ itemToDelete, itemType, onClose, onConfirm, currentTheme }) => {

const buttonStylePrimary = { backgroundColor: currentTheme.dangerColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50">

<div className="rounded-lg shadow-xl p-6 w-full max-w-md" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-xl font-bold mb-4" style={textColor}>Confirmar Exclusão</h2>

<p className="mb-6" style={secondaryTextColor}>

Tem certeza de que deseja excluir {itemType} "<span className="font-semibold">{itemToDelete.name || itemToDelete.atividade}</span>"? Esta ação não pode ser desfeita.

</p>

<div className="flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 text-base font-medium sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={onConfirm} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary}>Excluir</button>

</div>

</div>

</div>

);

});

// Componente para o Modal de Edição de Coluna Personalizada

const EditCustomColumnModal = React.memo(({ column, onClose, onSave, currentTheme }) => {

const [editedColumn, setEditedColumn] = useState(column);

const [error, setError] = useState('');

const inputStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const labelStyle = { color: currentTheme.secondaryTextColor };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const handleChange = useCallback((e) => {

const { name, value } = e.target;

setEditedColumn(prev => ({ ...prev, [name]: value }));

}, []);

const handleSave = useCallback(() => {

setError('');

if (!editedColumn.name.trim()) {

setError('O nome da coluna não pode ser vazio.');

return;

}

if (editedColumn.type === 'formula' && !editedColumn.formula.trim()) {

setError('A fórmula não pode ser vazia para colunas de fórmula.');

return;

}

onSave(editedColumn);

onClose();

}, [editedColumn, onClose, onSave]);

return (

<div className="fixed inset-0 bg-gray-600 bg-opacity-75 flex items-center justify-center p-4 z-50" style={{ backgroundColor: currentTheme.background }}>

<div className="rounded-lg shadow-xl p-6 w-full max-w-md max-h-[90vh] overflow-y-auto" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor }}>

<h2 className="text-2xl font-bold mb-4" style={{ color: currentTheme.textColor }}>Editar Coluna Personalizada</h2>

<div className="grid grid-cols-1 gap-4">

<div>

<label htmlFor="edit-column-name" className="block text-sm font-medium" style={labelStyle}>Nome da Coluna</label>

<input type="text" id="edit-column-name" name="name" value={editedColumn.name} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle} />

</div>

<div>

<label htmlFor="edit-column-type" className="block text-sm font-medium" style={labelStyle}>Tipo da Coluna</label>

<select id="edit-column-type" name="type" value={editedColumn.type} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle}>

<option value="text">Texto</option>

<option value="number">Número</option>

<option value="date">Data</option>

<option value="formula">Fórmula</option>

</select>

</div>

{editedColumn.type === 'formula' && (

<div>

<label htmlFor="edit-column-formula" className="block text-sm font-medium" style={labelStyle}>Fórmula</label>

<input type="text" id="edit-column-formula" name="formula" value={editedColumn.formula} onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300 shadow-sm sm:text-sm p-2" style={inputStyle} />

</div>

)}

</div>

{error && <p className="text-red-500 text-sm mt-4">{error}</p>}

<div className="mt-6 flex justify-end gap-3">

<button type="button" onClick={onClose} className="inline-flex justify-center rounded-md border border-gray-300 shadow-sm px-4 py-2 text-base font-medium sm:mt-0 sm:ml-3 sm:w-auto sm:text-sm" style={buttonStyleSecondary}>Cancelar</button>

<button type="button" onClick={handleSave} className="inline-flex justify-center rounded-md border border-transparent shadow-sm px-4 py-2 text-base font-medium text-white sm:ml-3 sm:w-auto sm:text-sm" style={buttonStylePrimary}>Salvar Alterações</button>

</div>

</div>

</div>

);

});

// Subcomponentes para o TableSettingsPanel

const ColumnVisibilitySettings = React.memo(({ visibleColumns, handleToggleColumnVisibility, currentTheme, allVisibleColumnsForSettings }) => {

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

return (

<>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Mostrar/Ocultar Colunas:</h3>

<div className="flex flex-wrap gap-x-6 gap-y-2 mb-6">

{allVisibleColumnsForSettings.map(col => (

<label key={col.key} className="inline-flex items-center text-sm" style={secondaryTextColor}>

<input type="checkbox" className="form-checkbox h-4 w-4 rounded" style={{ color: currentTheme.primaryColor }} checked={visibleColumns[col.key]} onChange={() => handleToggleColumnVisibility(col.key)} />

<span className="ml-2">{col.label}</span>

</label>

))}

</div>

</>

);

});

const CustomColumnManagement = React.memo(({

customColumns, setCustomColumns, newColumnName, setNewColumnName,

newColumnType, setNewColumnType, newColumnFormula, setNewColumnFormula,

handleAddColumn, handleRemoveColumn, handleEditCustomColumn, columnError, setColumnError, currentTheme,

handleToggleColumnVisibility, showFeedback

}) => {

const inputStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleDanger = { backgroundColor: currentTheme.dangerColor, color: currentTheme.textColor };

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

const handleMoveColumn = useCallback((index, direction) => {

setCustomColumns(prev => {

const newCols = [...prev];

const [movedColumn] = newCols.splice(index, 1);

const newIndex = index + direction;

if (newIndex >= 0 && newIndex < newCols.length + 1) {

newCols.splice(newIndex, 0, movedColumn);

}

return newCols;

});

showFeedback('Coluna movida!');

}, [setCustomColumns, showFeedback]);

return (

<>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Gerenciar Colunas Personalizadas:</h3>

<div className="flex flex-col gap-3 mb-4">

<input type="text" placeholder="Nome da Nova Coluna" className="p-2 border rounded-md" style={inputStyle} value={newColumnName} onChange={(e) => setNewColumnName(e.target.value)} />

<select value={newColumnType} onChange={(e) => setNewColumnType(e.target.value)} className="p-2 border rounded-md" style={inputStyle}>

<option value="text">Texto</option>

<option value="number">Número</option>

<option value="date">Data</option>

<option value="formula">Fórmula</option>

</select>

{newColumnType === 'formula' && (

<input type="text" placeholder="Fórmula (ex: valorReal - valorPlanejado)" className="p-2 border rounded-md" style={inputStyle} value={newColumnFormula} onChange={(e) => setNewColumnFormula(e.target.value)} />

)}

{columnError && <p className="text-red-500 text-sm">{columnError}</p>}

<button onClick={handleAddColumn} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={buttonStylePrimary}>Adicionar Coluna</button>

</div>

<div className="mb-6">

<h4 className="text-md font-semibold mb-2" style={textColor}>Colunas Personalizadas Atuais:</h4>

{customColumns.length === 0 ? (

<p className="text-sm italic" style={secondaryTextColor}>Nenhuma coluna personalizada adicionada ainda.</p>

) : (

<ul className="space-y-2">

{customColumns.map((col, index) => (

<li key={col.id} className="flex justify-between items-center p-2 rounded-md border" style={{ borderColor: currentTheme.tableBorder, backgroundColor: currentTheme.cardBackground }}>

<span className="font-medium" style={textColor}>{col.name} ({col.type}) {col.type === 'formula' && `: ${col.formula}`}</span>

<div className="flex items-center space-x-2">

<button onClick={() => handleEditCustomColumn(col)} className="px-2 py-1 rounded-md border text-xs transition-colors duration-200" style={{ borderColor: currentTheme.primaryColor, color: currentTheme.primaryColor }}>Editar</button>

<button onClick={() => handleRemoveColumn(col.id)} className="px-2 py-1 rounded-md border text-xs transition-colors duration-200" style={buttonStyleDanger}>Remover</button>

<button onClick={() => handleMoveColumn(index, -1)} disabled={index === 0} className="p-1 rounded-md text-xs" style={{ backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor }} title="Mover para Cima">

<svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" fill="none" viewBox="0 0 24 24" stroke="currentColor"><path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M5 10l7-7m0 0l7 7m-7-7v18" /></svg>

</button>

<button onClick={() => handleMoveColumn(index, 1)} disabled={index === customColumns.length - 1} className="p-1 rounded-md text-xs" style={{ backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor }} title="Mover para Baixo">

<svg xmlns="http://www.w3.org/2000/svg" className="h-4 w-4" fill="none" viewBox="0 0 24 24" stroke="currentColor"><path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M19 14l-7 7m0 0l-7-7m7 7V3" /></svg>

</button>

</div>

</li>

))}

</ul>

)}

</div>

</>

);

});

const StatusRiskManagement = React.memo(({

newStatusInput, setNewStatusInput, newStatusColor, setNewStatusColor, handleAddStatus, customStatuses,

newRiskInput, setNewRiskInput, newRiskColor, setNewRiskColor, handleAddRisk, customRisks,

currentTheme, getStatusTextColor, getRiskTextColor, getStatusBgColorInternal, getRiskBgColorInternal

}) => {

const inputStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const buttonStyleSecondary = { backgroundColor: currentTheme.secondaryColor, color: currentTheme.textColor };

const buttonStyleDanger = { backgroundColor: currentTheme.dangerColor, color: currentTheme.textColor };

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

return (

<>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Gerenciar Status:</h3>

<div className="flex flex-wrap gap-2 items-center mb-4">

<input type="text" placeholder="Novo Status" className="p-2 border rounded-md flex-grow max-w-xs" style={inputStyle} value={newStatusInput} onChange={(e) => setNewStatusInput(e.target.value)} />

<input type="color" value={newStatusColor} onChange={(e) => setNewStatusColor(e.target.value)} className="w-10 h-10 p-1 border rounded-md cursor-pointer" style={{ borderColor: currentTheme.tableBorder }} title="Escolher cor para o status" />

<button onClick={handleAddStatus} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={buttonStyleSecondary}>Adicionar Status</button>

</div>

<div className="flex flex-wrap gap-2 mb-6">

{customStatuses.map((status, index) => (

<span key={index} className="px-3 py-1 text-sm rounded-full" style={{ backgroundColor: getStatusBgColorInternal(status.name), color: getStatusTextColor(status.name) }}>

{status.name}

</span>

))}

</div>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Gerenciar Riscos:</h3>

<div className="flex flex-wrap gap-2 items-center mb-4">

<input type="text" placeholder="Novo Risco" className="p-2 border rounded-md flex-grow max-w-xs" style={inputStyle} value={newRiskInput} onChange={(e) => setNewRiskInput(e.target.value)} />

<input type="color" value={newRiskColor} onChange={(e) => setNewRiskColor(e.target.value)} className="w-10 h-10 p-1 border rounded-md cursor-pointer" style={{ borderColor: currentTheme.tableBorder }} title="Escolher cor para o risco" />

<button onClick={handleAddRisk} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={buttonStyleDanger}>Adicionar Risco</button>

</div>

<div className="flex flex-wrap gap-2">

{customRisks.map((risk, index) => (

<span key={index} className="px-3 py-1 text-sm rounded-full" style={{ backgroundColor: getRiskBgColorInternal(risk.name), color: getRiskTextColor(risk.name) }}>

{risk.name}

</span>

))}

</div>

</>

);

});

const CsvImportExportControls = React.memo(({

handleFileChange, isImportingCsv, handleExportCsv, csvFile, currentTheme, showFeedback

}) => {

const buttonStylePrimary = { backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor };

const buttonStyleSecondary = { backgroundColor: currentTheme.secondaryColor, color: currentTheme.textColor };

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

return (

<>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Importar Atividades (CSV):</h3>

<div className="flex flex-col sm:flex-row gap-3 items-center">

<input type="file" accept=".csv" onChange={handleFileChange} className="block w-full text-sm file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-blue-50 file:text-blue-700 hover:file:bg-blue-100" style={secondaryTextColor} />

<button onClick={() => {}} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200 flex items-center justify-center" style={buttonStylePrimary} disabled={isImportingCsv || !csvFile}>

{isImportingCsv ? (<><svg className="animate-spin -ml-1 mr-3 h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"><circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle><path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path></svg>Processando...</>) : ('Selecionar CSV')}

</button>

</div>

<p className="text-xs mt-2" style={secondaryTextColor}>

Certifique-se de que o CSV possui as colunas: atividade, disciplina, dataInicialPlanejada, dataFinalPlanejada, dataInicialReal, dataFinalReal, valorPlanejado, valorReal, statusPlanejado, statusReal, responsavel, prioridade, observacoes, dependencias, custoReal, recursosNecessarios, percentualConclusao, riscoAssociado, linkDocumento, dataUltimaAtualizacao.

</p>

<div className="mt-4">

<h3 className="text-lg font-semibold mb-3" style={textColor}>Exportar Atividades (CSV):</h3>

<button onClick={handleExportCsv} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200 flex items-center justify-center" style={buttonStyleSecondary}>

Exportar CSV

</button>

</div>

</>

);

});

// Componente para o Painel de Configurações da Tabela

const TableSettingsPanel = React.memo(({

visibleColumns, handleToggleColumnVisibility, feedbackMessage,

newStatusInput, setNewStatusInput, newStatusColor, setNewStatusColor, handleAddStatus, customStatuses,

newRiskInput, setNewRiskInput, newRiskColor, setNewRiskColor, handleAddRisk, customRisks,

handleFileChange, isImportingCsv, handleExportCsv, csvFile, currentTheme, getStatusTextColor, getRiskTextColor,

customColumns, setCustomColumns, allAvailableColumns, handleAddChartClick, showFeedback,

handleEditCustomColumn, columnError, setColumnError,

newColumnName, setNewColumnName, newColumnType, setNewColumnType, newColumnFormula, setNewColumnFormula // Added these props

}) => {

const textColor = { color: currentTheme.textColor };

const getStatusBgColorInternal = useCallback((statusName) => {

const defaultColors = { 'Concluído': '#D1FAE5', 'Em Andamento': '#DBEAFE', 'Atrasado': '#FEE2E2', 'Não Iniciado': '#F3F4F6' };

const custom = customStatuses.find(s => s.name === statusName);

return custom ? custom.color : defaultColors[statusName] || currentTheme.tableHeaderBg;

}, [customStatuses, currentTheme.tableHeaderBg]);

const getRiskBgColorInternal = useCallback((riskName) => {

const defaultColors = { 'Alto': '#FEE2E2', 'Média': '#FEF3C7', 'Baixo': '#D1FAE5' };

const custom = customRisks.find(r => r.name === riskName);

return custom ? custom.color : defaultColors[riskName] || currentTheme.tableHeaderBg;

}, [customRisks, currentTheme.tableHeaderBg]);

const allVisibleColumnsForSettings = useMemo(() => {

const standard = columnDefinitions.filter(col => col.key !== 'actions');

const custom = customColumns.map(col => ({ key: col.id, label: col.name, defaultVisible: true })); // Custom columns are visible by default

return [...standard, ...custom];

}, [customColumns]);

const handleAddColumn = useCallback(() => {

setColumnError('');

if (!newColumnName.trim()) {

setColumnError('O nome da coluna não pode ser vazio.');

return;

}

if (newColumnType === 'formula' && !newColumnFormula.trim()) {

setColumnError('A fórmula não pode ser vazia para colunas de fórmula.');

return;

}

const existingKeys = [...columnDefinitions.map(col => col.key), ...customColumns.map(col => col.id)];

const newColumnKey = newColumnName.trim().toLowerCase().replace(/\s+/g, '-'); // Simple slug for ID

if (existingKeys.includes(newColumnKey)) {

setColumnError('Já existe uma coluna com este nome (ou um nome similar).');

return;

}

const newCol = {

id: newColumnKey,

name: newColumnName.trim(),

type: newColumnType,

formula: newColumnType === 'formula' ? newColumnFormula.trim() : '',

};

setCustomColumns(prev => [...prev, newCol]);

handleToggleColumnVisibility(newCol.id); // Make it visible by default

setNewColumnName('');

setNewColumnType('text');

setNewColumnFormula('');

showFeedback(`Coluna "${newCol.name}" adicionada!`);

}, [newColumnName, newColumnType, newColumnFormula, customColumns, setCustomColumns, showFeedback, handleToggleColumnVisibility, setColumnError, setNewColumnName, setNewColumnType, setNewColumnFormula]);

const handleRemoveColumn = useCallback((columnId) => {

setCustomColumns(prev => prev.filter(col => col.id !== columnId));

showFeedback(`Coluna removida!`);

}, [setCustomColumns, showFeedback]);

return (

<div className="mb-6 p-4 rounded-lg shadow-inner transition-all duration-300 ease-in-out" style={{ backgroundColor: currentTheme.background }}>

{feedbackMessage && (

<div className="border-l-4 p-3 mb-4 rounded-md" style={{ backgroundColor: currentTheme.warningColor + '33', borderColor: currentTheme.warningColor, color: currentTheme.warningColor }}>

<p className="font-medium">{feedbackMessage}</p>

</div>

)}

<ColumnVisibilitySettings

visibleColumns={visibleColumns}

handleToggleColumnVisibility={handleToggleColumnVisibility}

currentTheme={currentTheme}

allVisibleColumnsForSettings={allVisibleColumnsForSettings}

/>

<CustomColumnManagement

customColumns={customColumns} setCustomColumns={setCustomColumns}

newColumnName={newColumnName} setNewColumnName={setNewColumnName}

newColumnType={newColumnType} setNewColumnType={setNewColumnType}

newColumnFormula={newColumnFormula} setNewColumnFormula={setNewColumnFormula}

columnError={columnError} setColumnError={setColumnError}

handleAddColumn={handleAddColumn}

handleRemoveColumn={handleRemoveColumn}

handleEditCustomColumn={handleEditCustomColumn}

handleToggleColumnVisibility={handleToggleColumnVisibility}

showFeedback={showFeedback}

currentTheme={currentTheme}

/>

<StatusRiskManagement

newStatusInput={newStatusInput} setNewStatusInput={setNewStatusInput}

newStatusColor={newStatusColor} setNewStatusColor={setNewStatusColor}

handleAddStatus={handleAddStatus} customStatuses={customStatuses}

newRiskInput={newRiskInput} setNewRiskInput={newRiskInput}

newRiskColor={newRiskColor} setNewRiskColor={newRiskColor}

handleAddRisk={handleAddRisk} customRisks={customRisks}

currentTheme={currentTheme}

getStatusTextColor={getStatusTextColor} getRiskTextColor={getRiskTextColor}

getStatusBgColorInternal={getStatusBgColorInternal} getRiskBgColorInternal={getRiskBgColorInternal}

/>

<CsvImportExportControls

handleFileChange={handleFileChange} isImportingCsv={isImportingCsv}

handleExportCsv={handleExportCsv} csvFile={csvFile}

currentTheme={currentTheme} showFeedback={showFeedback}

/>

<div className="mt-4">

<h3 className="text-lg font-semibold mb-3" style={textColor}>Criar Novo Gráfico:</h3>

<button onClick={handleAddChartClick} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200" style={{ backgroundColor: currentTheme.accentColor, color: currentTheme.textColor }}>Criar Novo Gráfico</button>

</div>

</div>

);

});

// Componente para a Seção de Relatório

const ReportGeneratorSection = React.memo(({

isReportSectionOpen, setIsReportSectionOpen, metrics, customCharts, currentTheme,

isGeneratingPdf, pdfLibsLoaded, handleExportPdf,

startDate, endDate, getStatusBgColor, getStatusTextColor

}) => {

const {

totalActivities, completedActivities, overdueActivities,

totalPlannedCost, totalRealCost, costDeviation,

statusChartData, completionPieData, costChartData, priorityChartData, riskChartData,

activitiesByResponsibleSummary, disciplineStatusData, responsibleStatusData, allPossibleStatuses,

activitiesWithCalculatedMetrics,

overallIndiceDesempenhoPrazo, overallVariacaoPrazo, overallIndiceDesempenhoCusto, overallVariacaoCusto

} = metrics;

const sectionStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor };

const subSectionStyle = { backgroundColor: currentTheme.background };

const textColor = { color: currentTheme.textColor };

const secondaryTextColor = { color: currentTheme.secondaryTextColor };

const tableHeaderStyle = (color) => ({ backgroundColor: color + '1A', color });

const tableBodyStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const chartAxisStyle = { stroke: currentTheme.secondaryTextColor };

const tooltipStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, border: `1px solid ${currentTheme.tableBorder}` };

const legendStyle = { color: currentTheme.textColor };

const gridStyle = { strokeDasharray: "3 3", stroke: currentTheme.tableBorder };

// Helper for KPI color based on value (repeated for report section)

const getKpiColor = (value, type) => {

if (type === 'index') { // SPI, CPI

return value >= 1 ? currentTheme.secondaryColor : currentTheme.dangerColor;

} else if (type === 'variance') { // SV, CV

return value >= 0 ? currentTheme.secondaryColor : currentTheme.dangerColor;

}

return currentTheme.textColor;

};

return (

<section id="report-content" className="p-6 rounded-lg shadow-md mt-8" style={sectionStyle}>

<h2 className="text-2xl font-bold mb-4" style={textColor}>Relatório de Atividades</h2>

<p className="mb-2" style={secondaryTextColor}>Gerado em: {new Date().toLocaleDateString('pt-BR')}</p>

<p className="mb-6" style={secondaryTextColor}>

Período do Relatório: {startDate && endDate ? `${formatDate(startDate)} a ${formatDate(endDate)}` : 'Todo o Período'}

</p>

<div className="flex justify-end mb-6 no-print">

<button onClick={handleExportPdf} className="px-4 py-2 rounded-md shadow-md transition-colors duration-200 flex items-center justify-center" style={{ backgroundColor: currentTheme.dangerColor, color: currentTheme.textColor }} disabled={isGeneratingPdf || !pdfLibsLoaded}>

{isGeneratingPdf ? (<><svg className="animate-spin -ml-1 mr-3 h-5 w-5 text-white" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24"><circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"></circle><path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"></path></svg>Gerando PDF...</>) : pdfLibsLoaded ? ('Exportar para PDF') : ('Carregando PDF...')}

</button>

</div>

{/\* Removida a seção de Análise de Observações com Gemini \*/}

<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 mb-8">

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.primaryColor + '1A', color: currentTheme.primaryColor }}><h3 className="text-lg font-semibold">Total de Atividades</h3><p className="text-2xl font-bold">{totalActivities}</p></div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.secondaryColor + '1A', color: currentTheme.secondaryColor }}><h3 className="text-lg font-semibold">Atividades Concluídas</h3><p className="text-2xl font-bold">{completedActivities}</p></div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.dangerColor + '1A', color: currentTheme.dangerColor }}><h3 className="text-lg font-semibold">Atividades Atrasadas</h3><p className="text-2xl font-bold">{overdueActivities.length}</p></div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.accentColor + '1A', color: currentTheme.accentColor }}><h3 className="text-lg font-semibold">Custo Total Real</h3><p className="text-2xl font-bold">{formatCurrency(totalRealCost)}</p></div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.warningColor + '1A', color: currentTheme.warningColor }}><h3 className="text-lg font-semibold">Custo Total Planejado</h3><p className="text-2xl font-bold">{formatCurrency(totalPlannedCost)}</p></div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: currentTheme.chartAccent + '1A', color: currentTheme.chartAccent }}><h3 className="text-lg font-semibold">Desvio de Custos</h3><p className="text-2xl font-bold">{formatCurrency(costDeviation)}</p></div>

{/\* New EVM KPIs in Report \*/}

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: getKpiColor(overallIndiceDesempenhoPrazo, 'index') + '1A', color: getKpiColor(overallIndiceDesempenhoPrazo, 'index') }}>

<h3 className="text-lg font-semibold">SPI</h3><p className="text-2xl font-bold">{overallIndiceDesempenhoPrazo.toFixed(2)}</p>

</div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: getKpiColor(overallVariacaoPrazo, 'variance') + '1A', color: getKpiColor(overallVariacaoPrazo, 'variance') }}>

<h3 className="text-lg font-semibold">SV</h3><p className="text-2xl font-bold">{formatCurrency(overallVariacaoPrazo)}</p>

</div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: getKpiColor(overallIndiceDesempenhoCusto, 'index') + '1A', color: getKpiColor(overallIndiceDesempenhoCusto, 'index') }}>

<h3 className="text-lg font-semibold">CPI</h3><p className="text-2xl font-bold">{overallIndiceDesempenhoCusto.toFixed(2)}</p>

</div>

<div className="p-4 rounded-lg shadow-sm" style={{ backgroundColor: getKpiColor(overallVariacaoCusto, 'variance') + '1A', color: getKpiColor(overallVariacaoCusto, 'variance') }}>

<h3 className="text-lg font-semibold">CV</h3><p className="text-2xl font-bold">{formatCurrency(overallVariacaoCusto)}</p>

</div>

</div>

<div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-8">

<div className="p-6 rounded-lg shadow-inner" style={subSectionStyle}>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Comparativo de Status (Planejado vs. Real)</h3>

<ResponsiveContainer width="100%" height={250}>

<BarChart data={statusChartData}><CartesianGrid strokeDasharray="3 3" stroke={currentTheme.tableBorder} /><XAxis dataKey="status" stroke={currentTheme.secondaryTextColor} /><YAxis stroke={currentTheme.secondaryTextColor} /><Tooltip contentStyle={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, border: `1px solid ${currentTheme.tableBorder}` }} /><Legend wrapperStyle={{ color: currentTheme.textColor }} /><Bar dataKey="Planejado" fill={currentTheme.chartPrimary} name="Planejado" radius={[5, 5, 0, 0]} /><Bar dataKey="Real" fill={currentTheme.chartSecondary} name="Real" radius={[5, 5, 0, 0]} /></BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-inner" style={subSectionStyle}>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Conclusão por Disciplina (%)</h3>

<ResponsiveContainer width="100%" height={250}>

<PieChart><Pie data={completionPieData} cx="50%" cy="50%" labelLine={false} outerRadius={80} fill={currentTheme.chartPrimary} dataKey="value" label={({ name, percent }) => `${name}: ${(percent \* 100).toFixed(0)}%`} >{completionPieData.map((entry, index) => (<Cell key={`cell-${index}`} fill={PIE\_COLORS[index % PIE\_COLORS.length]} />))}</Pie><Tooltip formatter={(value) => `${value.toFixed(1)}%`} contentStyle={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, border: `1px solid ${currentTheme.tableBorder}` }} /><Legend wrapperStyle={{ color: currentTheme.textColor }} /></PieChart>

</ResponsiveContainer>

</div>

{/\* Novos Gráficos Cruzados no Relatório \*/}

<div className="p-6 rounded-lg shadow-inner" style={subSectionStyle}>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Atividades por Disciplina e Status</h3>

<ResponsiveContainer width="100%" height={250}>

<BarChart data={disciplineStatusData}><CartesianGrid style={gridStyle} /><XAxis dataKey="disciplina" style={chartAxisStyle} /><YAxis style={chartAxisStyle} /><Tooltip contentStyle={tooltipStyle} /><Legend wrapperStyle={legendStyle} />

{allPossibleStatuses.map(status => (

<Bar key={status} dataKey={status} stackId="a" fill={getStatusBgColor(status)} name={status} />

))}

</BarChart>

</ResponsiveContainer>

</div>

<div className="p-6 rounded-lg shadow-inner" style={subSectionStyle}>

<h3 className="text-lg font-semibold mb-3" style={textColor}>Atividades por Responsável e Status</h3>

<ResponsiveContainer width="100%" height={250}>

<BarChart data={responsibleStatusData}><CartesianGrid style={gridStyle} /><XAxis dataKey="responsavel" style={chartAxisStyle} /><YAxis style={chartAxisStyle} /><Tooltip contentStyle={tooltipStyle} /><Legend wrapperStyle={legendStyle} />

{allPossibleStatuses.map(status => (

<Bar key={status} dataKey={status} stackId="a" fill={getStatusBgColor(status)} name={status} />

))}

</BarChart>

</ResponsiveContainer>

</div>

{/\* Gráficos personalizados no relatório \*/}

{customCharts.map(chart => (

<CustomChartDisplay

key={chart.id}

chart={chart}

data={activitiesWithCalculatedMetrics}

currentTheme={currentTheme}

getStatusBgColor={getStatusBgColor}

/>

))}

</div>

<h3 className="text-xl font-semibold mb-4" style={textColor}>Resumo de Custos por Disciplina</h3>

{costChartData.length > 0 ? (<div className="overflow-x-auto rounded-lg border mb-8" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.primaryColor)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Disciplina</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Valor Planejado</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Valor Real</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Desvio</th></tr></thead><tbody style={tableBodyStyle}>{costChartData.map((data, index) => (<tr key={index}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{data.disciplina}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{formatCurrency(data.Planejado)}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{formatCurrency(data.Real)}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: data.Real - data.Planejado > 0 ? currentTheme.dangerColor : currentTheme.secondaryColor }}>{formatCurrency(data.Real - data.Planejado)}</td></tr>))}</tbody></table></div>) : (<p className="mb-8" style={secondaryTextColor}>Nenhum resumo de custos por disciplina disponível.</p>)}

<h3 className="text-xl font-semibold mb-4" style={textColor}>Resumo de Atividades por Prioridade</h3>

{priorityChartData.length > 0 ? (<div className="overflow-x-auto rounded-lg border mb-8" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.chartPriority)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.chartPriority }}>Prioridade</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.chartPriority }}>Número de Atividades</th></tr></thead><tbody style={tableBodyStyle}>{priorityChartData.map((data, index) => (<tr key={index}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{data.name}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{data.count}</td></tr>))}</tbody></table></div>) : (<p className="mb-8" style={secondaryTextColor}>Nenhum resumo de atividades por prioridade disponível.</p>)}

<h3 className="text-xl font-semibold mb-4" style={textColor}>Resumo de Atividades por Risco</h3>

{riskChartData.length > 0 ? (<div className="overflow-x-auto rounded-lg border mb-8" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.chartRisk)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.chartRisk }}>Risco</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.chartRisk }}>Número de Atividades</th></tr></thead><tbody style={tableBodyStyle}>{riskChartData.map((data, index) => (<tr key={index}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{data.name}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{data.count}</td></tr>))}</tbody></table></div>) : (<p className="mb-8" style={secondaryTextColor}>Nenhum resumo de atividades por risco disponível.</p>)}

<h3 className="text-xl font-semibold mb-4" style={textColor}>Atividades Atrasadas</h3>

{overdueActivities.length > 0 ? (<div className="overflow-x-auto rounded-lg border mb-8" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.dangerColor)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.dangerColor }}>Atividade</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.dangerColor }}>Responsável</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.dangerColor }}>Data Final Planejada</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.dangerColor }}>Status Real</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.dangerColor }}>Dias Atraso</th></tr></thead><tbody style={tableBodyStyle}>{overdueActivities.map(activity => (<tr key={activity.id}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{activity.atividade}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{activity.responsavel}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{formatDate(activity.dataFinalPlanejada)}</td><td className="px-6 py-4 whitespace-nowrap text-sm"><span className="px-2 inline-flex text-xs leading-5 font-semibold rounded-full" style={{ backgroundColor: getStatusBgColor(activity.statusReal), color: getStatusTextColor(activity.statusReal) }}>{activity.statusReal}</span></td><td className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.dangerColor }}>{activity.diasAtraso}</td></tr>))}</tbody></table></div>) : (<p className="mb-8" style={secondaryTextColor}>Nenhuma atividade atrasada encontrada.</p>)}

<h3 className="text-xl font-semibold mb-4" style={textColor}>Atividades com Dias Remanescentes</h3>

{metrics.remainingDaysActivities.length > 0 ? (<div className="overflow-x-auto rounded-lg border mb-8" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.secondaryColor)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryColor }}>Atividade</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryColor }}>Responsável</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryColor }}>Data Final Planejada</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryColor }}>Status Real</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.secondaryColor }}>Dias Remanescentes</th></tr></thead><tbody style={tableBodyStyle}>{metrics.remainingDaysActivities.map(activity => (<tr key={activity.id}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{activity.atividade}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{activity.responsavel}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{formatDate(activity.dataFinalPlanejada)}</td><td className="px-6 py-4 whitespace-nowrap text-sm"><span className="px-2 inline-flex text-xs leading-5 font-semibold rounded-full" style={{ backgroundColor: getStatusBgColor(activity.statusReal), color: getStatusTextColor(activity.statusReal) }}>{activity.statusReal}</span></td><td className="px-6 py-4 whitespace-nowrap text-sm" style={{ color: currentTheme.secondaryColor }}>{activity.diasRemanescentes}</td></tr>))}</tbody></table></div>) : (<p className="mb-8" style={secondaryTextColor}>Nenhuma atividade com dias remanescentes encontrada.</p>)}

<h3 className="text-xl font-semibold mb-4" style={textColor}>Resumo de Atividades por Responsável</h3>

{Object.keys(activitiesByResponsibleSummary).length > 0 ? (<div className="overflow-x-auto rounded-lg border" style={{ borderColor: currentTheme.tableBorder }}><table className="min-w-full divide-y" style={{ borderColor: currentTheme.tableBorder }}><thead style={tableHeaderStyle(currentTheme.primaryColor)}><tr ><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Responsável</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Total</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Concluído</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Em Andamento</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Atrasado</th><th className="px-6 py-3 text-left text-xs font-medium uppercase tracking-wider" style={{ color: currentTheme.primaryColor }}>Não Iniciado</th></tr></thead><tbody style={tableBodyStyle}>{Object.entries(activitiesByResponsibleSummary).map(([responsavel, summary]) => (<tr key={responsavel}><td className="px-6 py-4 whitespace-nowrap text-sm font-medium" style={textColor}>{responsavel}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{summary.total}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{summary.concluido}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{summary.emAndamento}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{summary.atrasado}</td><td className="px-6 py-4 whitespace-nowrap text-sm" style={secondaryTextColor}>{summary.naoIniciado}</td></tr>))}</tbody></table></div>) : (<p className="" style={secondaryTextColor}>Nenhum resumo por responsável disponível.</p>)}

</section>

);

});

// New Component: Gantt Chart

const GanttChart = React.memo(({ activities, currentTheme, setAllActivities, showFeedback }) => {

const chartRef = useRef(null);

const [scale, setScale] = useState(1); // Days per pixel

const [minDate, setMinDate] = useState(null);

const [maxDate, setMaxDate] = useState(null);

const headerStyle = { backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor };

const taskRowStyle = { backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder };

const baselineBarColor = currentTheme.primaryColor + '40'; // Lighter primary color for baseline

const actualBarColor = currentTheme.primaryColor; // Primary color for actual

const todayLineColor = currentTheme.dangerColor;

useEffect(() => {

if (activities.length > 0) {

const allDates = activities.flatMap(activity => [

activity.dataInicialPlanejada, activity.dataFinalPlanejada,

activity.dataInicialReal, activity.dataFinalReal,

activity.dataInicialBaseline, activity.dataFinalBaseline

].filter(Boolean).map(d => new Date(d)));

if (allDates.length > 0) {

const min = new Date(Math.min(...allDates));

const max = new Date(Math.max(...allDates));

setMinDate(min);

setMaxDate(max);

}

}

}, [activities]);

const calculatePositionAndWidth = useCallback((startDate, endDate, chartWidth) => {

if (!startDate || !endDate || !minDate || !maxDate || chartWidth === 0) {

return { left: 0, width: 0 };

}

const start = new Date(startDate);

const end = new Date(endDate);

const totalDurationMs = maxDate.getTime() - minDate.getTime();

if (totalDurationMs === 0) return { left: 0, width: 0 }; // Avoid division by zero

const startOffsetMs = start.getTime() - minDate.getTime();

const taskDurationMs = end.getTime() - start.getTime();

const left = (startOffsetMs / totalDurationMs) \* chartWidth;

const width = (taskDurationMs / totalDurationMs) \* chartWidth;

return { left, width: Math.max(0, width) }; // Ensure width is not negative

}, [minDate, maxDate]);

const setBaseline = useCallback(() => {

setAllActivities(prevActivities =>

prevActivities.map(activity => ({

...activity,

dataInicialBaseline: activity.dataInicialPlanejada,

dataFinalBaseline: activity.dataFinalPlanejada,

}))

);

showFeedback('Linha de base definida com sucesso!');

}, [setAllActivities, showFeedback]);

const clearBaseline = useCallback(() => {

setAllActivities(prevActivities =>

prevActivities.map(activity => ({

...activity,

dataInicialBaseline: '',

dataFinalBaseline: '',

}))

);

showFeedback('Linha de base limpa!');

}, [setAllActivities, showFeedback]);

const todayPosition = useMemo(() => {

if (!minDate || !maxDate || !chartRef.current) return -1;

const chartWidth = chartRef.current.offsetWidth;

const today = new Date();

today.setHours(0, 0, 0, 0); // Normalize to start of day

const totalDurationMs = maxDate.getTime() - minDate.getTime();

if (totalDurationMs === 0) return -1;

const todayOffsetMs = today.getTime() - minDate.getTime();

return (todayOffsetMs / totalDurationMs) \* chartWidth;

}, [minDate, maxDate, activities]); // Re-calculate if activities change (dates might shift)

return (

<section className="p-6 rounded-lg shadow-md mb-8" style={headerStyle}>

<h2 className="text-xl font-semibold mb-4" style={{ color: currentTheme.textColor }}>Gráfico de Gantt</h2>

<div className="flex justify-end gap-3 mb-4">

<button onClick={setBaseline} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor }}>Definir Linha de Base</button>

<button onClick={clearBaseline} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.dangerColor, color: currentTheme.textColor }}>Limpar Linha de Base</button>

</div>

<div className="overflow-x-auto rounded-lg border" style={{ borderColor: currentTheme.tableBorder }}>

<div className="min-w-[800px] relative" ref={chartRef}> {/\* Min-width for better visualization \*/}

{minDate && maxDate && (

<div className="flex justify-between text-xs p-2 font-semibold" style={{ backgroundColor: currentTheme.tableHeaderBg, color: currentTheme.secondaryTextColor }}>

<span>{formatDate(minDate)}</span>

<span>{formatDate(maxDate)}</span>

</div>

)}

{todayPosition !== -1 && (

<div className="absolute top-0 bottom-0 w-0.5 z-10" style={{ left: `${todayPosition}px`, backgroundColor: todayLineColor }}>

<span className="absolute -top-6 left-1/2 -translate-x-1/2 text-xs font-bold whitespace-nowrap" style={{ color: todayLineColor }}>Hoje</span>

</div>

)}

<div className="divide-y" style={{ borderColor: currentTheme.tableBorder }}>

{activities.map(activity => {

const planned = calculatePositionAndWidth(activity.dataInicialPlanejada, activity.dataFinalPlanejada, chartRef.current?.offsetWidth || 0);

const actual = calculatePositionAndWidth(activity.dataInicialReal, activity.dataFinalReal, chartRef.current?.offsetWidth || 0);

const baseline = calculatePositionAndWidth(activity.dataInicialBaseline, activity.dataFinalBaseline, chartRef.current?.offsetWidth || 0);

return (

<div key={activity.id} className="flex items-center py-3 px-2 relative" style={taskRowStyle}>

<div className="w-40 text-sm font-medium truncate" style={{ color: currentTheme.textColor }}>{activity.atividade}</div>

<div className="flex-1 h-8 relative ml-4">

{baseline.width > 0 && (

<div

className="absolute h-4 rounded-md opacity-70"

style={{

left: `${baseline.left}px`,

width: `${baseline.width}px`,

backgroundColor: baselineBarColor,

top: '50%',

transform: 'translateY(-50%)'

}}

title={`Linha de Base: ${formatDate(activity.dataInicialBaseline)} - ${formatDate(activity.dataFinalBaseline)}`}

></div>

)}

{actual.width > 0 && (

<div

className="absolute h-4 rounded-md"

style={{

left: `${actual.left}px`,

width: `${actual.width}px`,

backgroundColor: actualBarColor,

top: '50%',

transform: 'translateY(-50%)'

}}

title={`Real: ${formatDate(activity.dataInicialReal)} - ${formatDate(activity.dataFinalReal)}`}

></div>

)}

</div>

</div>

);

})}

</div>

</div>

</div>

<p className="text-sm mt-4" style={{ color: currentTheme.secondaryTextColor }}>

Barras mais claras representam a linha de base (planejamento original). Barras mais escuras representam o progresso real.

</p>

</section>

);

});

// New Component: Kanban Board

const KanbanBoard = React.memo(({ activities, onUpdateActivityStatus, customStatuses, currentTheme, getStatusBgColor, getStatusTextColor }) => {

const defaultStatuses = ['Não Iniciado', 'Em Andamento', 'Concluído', 'Atrasado'];

const allKanbanStatuses = useMemo(() => [...new Set([...defaultStatuses, ...customStatuses.map(s => s.name)])], [customStatuses]);

const handleDragStart = useCallback((e, activityId) => {

e.dataTransfer.setData("activityId", activityId);

}, []);

const handleDragOver = useCallback((e) => {

e.preventDefault(); // Necessary to allow dropping

}, []);

const handleDrop = useCallback((e, newStatus) => {

e.preventDefault();

const activityId = e.dataTransfer.getData("activityId");

const activity = activities.find(a => a.id === parseFloat(activityId));

if (activity && activity.statusReal !== newStatus) {

onUpdateActivityStatus(activity.id, newStatus);

}

}, [activities, onUpdateActivityStatus]);

const columnStyle = { backgroundColor: currentTheme.background, borderColor: currentTheme.tableBorder };

const cardStyle = { backgroundColor: currentTheme.cardBackground, borderColor: currentTheme.tableBorder, color: currentTheme.textColor };

const columnHeaderStyle = { color: currentTheme.textColor };

const cardSecondaryTextStyle = { color: currentTheme.secondaryTextColor };

return (

<section className="p-6 rounded-lg shadow-md mb-8" style={{ backgroundColor: currentTheme.cardBackground }}>

<h2 className="text-xl font-semibold mb-4" style={{ color: currentTheme.textColor }}>Quadro Kanban</h2>

<div className="flex overflow-x-auto space-x-4 pb-4">

{allKanbanStatuses.map(status => (

<div

key={status}

className="flex-shrink-0 w-80 p-4 rounded-lg shadow-inner border"

style={columnStyle}

onDragOver={handleDragOver}

onDrop={(e) => handleDrop(e, status)}

>

<h3 className="text-lg font-semibold mb-4 text-center" style={columnHeaderStyle}>{status}</h3>

<div className="space-y-3">

{activities.filter(activity => activity.statusReal === status).map(activity => (

<div

key={activity.id}

draggable

onDragStart={(e) => handleDragStart(e, activity.id)}

className="p-3 rounded-md shadow-sm border cursor-grab active:cursor-grabbing"

style={cardStyle}

>

<p className="font-medium text-sm mb-1">{activity.atividade}</p>

<p className="text-xs" style={cardSecondaryTextStyle}>Responsável: {activity.responsavel}</p>

<p className="text-xs" style={cardSecondaryTextStyle}>Disciplina: {activity.disciplina}</p>

<p className="text-xs" style={cardSecondaryTextStyle}>Conclusão: {activity.percentualConclusao}%</p>

{activity.diasAtraso > 0 && (

<p className="text-xs font-semibold mt-1" style={{ color: currentTheme.dangerColor }}>

Atrasado: {activity.diasAtraso} dias

</p>

)}

</div>

))}

</div>

</div>

))}

</div>

<p className="text-sm mt-4" style={{ color: currentTheme.secondaryTextColor }}>

Arraste e solte os cartões entre as colunas para atualizar o status das atividades.

</p>

</section>

);

});

// --- COMPONENTE PRINCIPAL ---

const App = () => {

const {

allActivities, setAllActivities, customStatuses, setCustomStatuses, customRisks, setCustomRisks,

customColumns, setCustomColumns, customCharts, setCustomCharts,

userId, isAuthReady, activeDashboardId, setActiveDashboardId, activeDashboardName, setActiveDashboardName

} = useDashboardData();

const [showDashboardSelector, setShowDashboardSelector] = useState(true);

const [currentThemeKey, setCurrentThemeKeys] = useState('light'); // Changed to setCurrentThemeKeys to avoid conflict

const currentTheme = themes[currentThemeKey];

const [feedbackMessage, setFeedbackMessage] = useState('');

const [showFeedbackToast, setShowFeedbackToast] = useState(false);

// Função para mostrar feedback temporário

const showFeedback = useCallback((message) => {

setFeedbackMessage(message);

setShowFeedbackToast(true);

const timer = setTimeout(() => {

setShowFeedbackToast(false);

setFeedbackMessage('');

}, 3000);

return () => clearTimeout(timer);

}, []);

// Estados para filtros e modais

const [searchTerm, setSearchTerm] = useState('');

const [filterStatus, setFilterStatus] = useState('Todos');

const [filterResponsavel, setFilterResponsavel] = useState('Todos');

const [startDate, setStartDate] = useState('');

const [endDate, setEndDate] = useState('');

const [selectedPeriod, setSelectedPeriod] = useState('Todos');

const [isEditModalOpen, setIsEditModalOpen] = useState(false);

const [currentActivityToEdit, setCurrentActivityToEdit] = useState(null);

const [isAddModalOpen, setIsAddModalOpen] = useState(false);

const [isTableSettingsOpen, setIsTableSettingsOpen] = useState(false);

const [isReportSectionOpen, setIsReportSectionOpen] = useState(false);

const [isCreateChartModalOpen, setIsCreateChartModalOpen] = useState(false); // Novo estado para modal de criação de gráfico

const [isDeleteConfirmModalOpen, setIsDeleteConfirmModalOpen] = useState(false);

const [itemToDelete, setItemToDelete] = useState(null);

const [itemTypeToDelete, setItemTypeToDelete] = useState('');

const [isEditCustomColumnModalOpen, setIsEditCustomColumnModalOpen] = useState(false);

const [currentCustomColumnToEdit, setCurrentCustomColumnToEdit] = useState(null);

// New states for Gantt and Kanban visibility

const [showGanttChart, setShowGanttChart] = useState(false);

const [showKanbanBoard, setShowKanbanBoard] = useState(false);

// Hook para métricas calculadas

const metrics = useActivityMetrics(allActivities, customStatuses, customRisks, customColumns, searchTerm, filterStatus, filterResponsavel, startDate, endDate);

// Hook para importação/exportação CSV

const {

csvFile, isImportingCsv, csvPreviewData, csvPreviewHeaders, showCsvPreviewModal,

handleFileChange, handleConfirmImport, handleCancelImport, handleExportCsv

} = useCsvImportExport(setAllActivities, showFeedback, metrics.activitiesWithCalculatedMetrics, columnDefinitions, customColumns);

// Hook para relatório PDF

const { isGeneratingPdf, pdfLibsLoaded, handleExportPdf } = usePdfReport(showFeedback);

// Estados para gerenciamento de status/riscos personalizados (movidos para o App para serem passados para TableSettingsPanel)

const [newStatusInput, setNewStatusInput] = useState('');

const [newStatusColor, setNewStatusColor] = useState('#000000');

const [newRiskInput, setNewRiskInput] = useState('');

const [newRiskColor, setNewRiskColor] = useState('#000000');

const [columnError, setColumnError] = useState('');

// Estados para gerenciamento de novas colunas personalizadas (movidos para o App)

const [newColumnName, setNewColumnName] = useState('');

const [newColumnType, setNewColumnType] = useState('text');

const [newColumnFormula, setNewColumnFormula] = useState('');

// Funções para gerenciar status/riscos (passadas para TableSettingsPanel)

const handleAddStatus = useCallback(() => {

const trimmedInput = newStatusInput.trim();

const defaultStatuses = ['Concluído', 'Em Andamento', 'Atrasado', 'Não Iniciado'];

const allExistingStatusNames = [...new Set([...defaultStatuses, ...customStatuses.map(s => s.name)])];

if (trimmedInput === '') {

showFeedback('O nome do status não pode ser vazio!');

return;

}

if (allExistingStatusNames.includes(trimmedInput)) {

showFeedback('Status já existe!');

return;

}

setCustomStatuses(prev => [...prev, { name: trimmedInput, color: newStatusColor }]);

setNewStatusInput('');

setNewStatusColor('#000000');

showFeedback(`Status "${trimmedInput}" adicionado!`);

}, [newStatusInput, newStatusColor, customStatuses, setCustomStatuses, showFeedback]);

const handleAddRisk = useCallback(() => {

const trimmedInput = newRiskInput.trim();

const defaultRisks = ['Alto', 'Média', 'Baixo'];

const allExistingRiskNames = [...new Set([...defaultRisks, ...customRisks.map(r => r.name)])];

if (trimmedInput === '') {

showFeedback('O nome do risco não pode ser vazio!');

return;

}

if (allExistingRiskNames.includes(trimmedInput)) {

showFeedback('Risco já existe!');

return;

}

setCustomRisks(prev => [...prev, { name: trimmedInput, color: newRiskColor }]);

setNewRiskInput('');

setNewRiskColor('#000000');

showFeedback(`Risco "${trimmedInput}" adicionado!`);

}, [newRiskInput, newRiskColor, customRisks, setCustomRisks, showFeedback]);

// Função para obter cor de fundo do status

const getStatusBgColor = useCallback((statusName) => {

const defaultColors = { 'Concluído': '#D1FAE5', 'Em Andamento': '#DBEAFE', 'Atrasado': '#FEE2E2', 'Não Iniciado': '#F3F4F6' };

const custom = customStatuses.find(s => s.name === statusName);

return custom ? custom.color : defaultColors[statusName] || currentTheme.tableHeaderBg;

}, [customStatuses, currentTheme.tableHeaderBg]);

// Função para obter cor do texto do status

const getStatusTextColor = useCallback((statusName) => {

const defaultTextColors = { 'Concluído': '#065F46', 'Em Andamento': '#1E40AF', 'Atrasado': '#991B1B', 'Não Iniciado': '#374151' };

const custom = customStatuses.find(s => s.name === statusName);

if (custom) { return getLuminance(custom.color) > 0.5 ? '#1F2937' : '#FFFFFF'; }

return defaultTextColors[statusName] || currentTheme.textColor;

}, [customStatuses, currentTheme.textColor]);

// Função para obter cor de fundo do risco

const getRiskBgColor = useCallback((riskName) => {

const defaultColors = { 'Alto': '#FEE2E2', 'Média': '#FEF3C7', 'Baixo': '#D1FAE5' };

const custom = customRisks.find(r => r.name === riskName);

return custom ? custom.color : defaultColors[riskName] || currentTheme.tableHeaderBg;

}, [customRisks, currentTheme.tableHeaderBg]);

// Função para obter cor do texto do risco

const getRiskTextColor = useCallback((riskName) => {

const defaultTextColors = { 'Alto': '#991B1B', 'Média': '#92400E', 'Baixo': '#065F46' };

const custom = customRisks.find(r => r.name === riskName);

if (custom) { return getLuminance(custom.color) > 0.5 ? '#1F2937' : '#FFFFFF'; }

return defaultTextColors[riskName] || currentTheme.textColor;

}, [customRisks, currentTheme.textColor]);

// Gerenciamento de visibilidade de colunas

const [visibleColumns, setVisibleColumns] = useState(() => {

const initialVisibility = {};

columnDefinitions.forEach(col => { initialVisibility[col.key] = col.defaultVisible; });

return initialVisibility;

});

const handleToggleColumnVisibility = useCallback((key) => {

setVisibleColumns(prev => ({ ...prev, [key]: !prev[key] }));

}, []);

// Callbacks para modais de edição/adição

const handleEditClick = useCallback((activity) => {

setCurrentActivityToEdit(activity);

setIsEditModalOpen(true);

}, []);

const handleCloseEditModal = useCallback(() => {

setIsEditModalOpen(false);

setCurrentActivityToEdit(null);

}, []);

const handleSaveActivity = useCallback((updatedActivity) => {

setAllActivities(prevActivities =>

prevActivities.map(act => (act.id === updatedActivity.id ? updatedActivity : act))

);

}, [setAllActivities]);

const handleAddClick = useCallback(() => {

setIsAddModalOpen(true);

}, []);

const handleCloseAddModal = useCallback(() => {

setIsAddModalOpen(false);

}, []);

const handleAddNewActivity = useCallback((newActivity) => {

setAllActivities(prevActivities => [...prevActivities, newActivity]);

}, [setAllActivities]);

// Callbacks para gráficos personalizados

const handleAddChartClick = useCallback(() => {

setIsCreateChartModalOpen(true);

}, []);

const handleCloseCreateChartModal = useCallback(() => {

setIsCreateChartModalOpen(false);

}, []);

const handleSaveCustomChart = useCallback((newChart) => {

setCustomCharts(prev => [...prev, newChart]);

showFeedback(`Gráfico "${newChart.name}" adicionado!`);

}, [setCustomCharts, showFeedback]);

// Funções para exclusão de atividade

const handleDeleteClick = useCallback((activity) => {

setItemToDelete(activity);

setItemTypeToDelete('atividade');

setIsDeleteConfirmModalOpen(true);

}, []);

const handleConfirmDelete = useCallback(() => {

if (itemTypeToDelete === 'atividade') {

setAllActivities(prevActivities => prevActivities.filter(act => act.id !== itemToDelete.id));

showFeedback(`Atividade "${itemToDelete.atividade}" excluída com sucesso!`);

} else if (itemTypeToDelete === 'coluna personalizada') {

setCustomColumns(prev => prev.filter(col => col.id !== itemToDelete.id));

showFeedback(`Coluna "${itemToDelete.name}" excluída com sucesso!`);

}

setIsDeleteConfirmModalOpen(false);

setItemToDelete(null);

setItemTypeToDelete('');

}, [itemToDelete, itemTypeToDelete, setAllActivities, showFeedback, setCustomColumns]);

const handleCancelDelete = useCallback(() => {

setIsDeleteConfirmModalOpen(false);

setItemToDelete(null);

setItemTypeToDelete('');

}, []);

// Funções para edição de coluna personalizada

const handleEditCustomColumn = useCallback((column) => {

setCurrentCustomColumnToEdit(column);

setIsEditCustomColumnModalOpen(true);

}, []);

const handleCloseEditCustomColumnModal = useCallback(() => {

setIsEditCustomColumnModalOpen(false);

setCurrentCustomColumnToEdit(null);

}, []);

const handleSaveEditedCustomColumn = useCallback((updatedColumn) => {

setCustomColumns(prev => prev.map(col => col.id === updatedColumn.id ? updatedColumn : col));

showFeedback(`Coluna "${updatedColumn.name}" atualizada!`);

}, [setCustomColumns, showFeedback]);

// Callback for Kanban board to update activity status

const handleUpdateActivityStatus = useCallback((activityId, newStatus) => {

setAllActivities(prevActivities =>

prevActivities.map(act =>

act.id === activityId ? { ...act, statusReal: newStatus } : act

)

);

showFeedback(`Status da atividade atualizado para "${newStatus}"!`);

}, [setAllActivities, showFeedback]);

if (!isAuthReady) {

return (

<div className="min-h-screen flex items-center justify-center" style={{ backgroundColor: currentTheme.background, color: currentTheme.textColor }}>

<p>Inicializando Firebase e autenticando...</p>

</div>

);

}

if (showDashboardSelector) {

return (

<DashboardSelectorModal

currentTheme={currentTheme}

onSelectDashboard={(id) => { setActiveDashboardId(id); setShowDashboardSelector(false); }}

onClose={() => { /\* Lógica para fechar se necessário \*/ }}

/>

);

}

return (

<div className="min-h-screen p-4 font-inter max-w-7xl mx-auto" style={{ backgroundColor: currentTheme.background, color: currentTheme.textColor }}>

<header className="shadow rounded-lg p-6 mb-6 no-print" style={{ backgroundColor: currentTheme.cardBackground }}>

<div className="flex flex-col sm:flex-row justify-between items-start sm:items-center">

<h1 className="text-2xl sm:text-3xl font-bold mb-2 sm:mb-0" style={{ color: currentTheme.textColor }}>Dashboard de Atividades: {activeDashboardName}</h1>

<div className="flex flex-col sm:flex-row items-stretch sm:items-center space-y-2 sm:space-y-0 sm:space-x-2 w-full sm:w-auto">

<button onClick={() => setShowDashboardSelector(true)} className="px-4 py-2 rounded-md shadow-sm transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor }}>Mudar Dashboard</button>

<label htmlFor="theme-select" className="text-sm font-medium" style={{ color: currentTheme.secondaryTextColor }}>Tema:</label>

<select id="theme-select" className="p-2 border rounded-md text-sm" style={{ backgroundColor: currentTheme.cardBackground, color: currentTheme.textColor, borderColor: currentTheme.tableBorder }} value={currentThemeKey} onChange={(e) => setCurrentThemeKeys(e.target.value)}>

{Object.keys(themes).map(key => (<option key={key} value={key}>{themes[key].name}</option>))}

</select>

</div>

</div>

<p className="mt-2 text-sm sm:text-base" style={{ color: currentTheme.secondaryTextColor }}>Acompanhe o progresso das suas atividades e compare o desempenho real com o planejado.</p>

<p className="text-xs mt-2" style={{ color: currentTheme.secondaryTextColor }}>ID do Usuário: {userId}</p>

{/\* Global Search Bar \*/}

<div className="mt-4">

<input type="text" placeholder="Pesquisa global..." className="p-3 border rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 w-full" style={{ backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder }} value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} />

</div>

</header>

{showFeedbackToast && feedbackMessage && (

<div className="fixed top-4 right-4 z-50 bg-green-500 text-white p-3 rounded-md shadow-lg transition-opacity duration-300" style={{ backgroundColor: currentTheme.secondaryColor }}>

{feedbackMessage}

</div>

)}

<OverviewCards metrics={metrics} currentTheme={currentTheme} />

<DateFilterSection startDate={startDate} setStartDate={setStartDate} endDate={endDate} setEndDate={setEndDate} selectedPeriod={selectedPeriod} setSelectedPeriod={setSelectedPeriod} currentTheme={currentTheme} />

<ChartsDisplaySection metrics={metrics} customCharts={customCharts} currentTheme={currentTheme} getStatusBgColor={getStatusBgColor} />

{/\* Seção da Tabela de Atividades - Controles e Tabela \*/}

<section className="p-6 rounded-lg shadow-md no-print" style={{ backgroundColor: currentTheme.cardBackground }}>

<div className="flex flex-col sm:flex-row justify-between items-start sm:items-center mb-6">

<h2 className="text-2xl font-bold mb-4 sm:mb-0" style={{ color: currentTheme.textColor }}>Detalhes das Atividades</h2>

<div className="flex flex-col sm:flex-row items-stretch sm:items-center space-y-2 sm:space-y-0 sm:space-x-3 w-full sm:w-auto">

<button onClick={() => setIsTableSettingsOpen(!isTableSettingsOpen)} className="p-2 rounded-full shadow-md transition-colors duration-200" style={{ backgroundColor: currentTheme.background, color: currentTheme.textColor, borderColor: currentTheme.tableBorder }} title="Personalizar Colunas da Tabela">

<svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor" strokeWidth="2"><path strokeLinecap="round" strokeLinejoin="round" d="M10.325 4.317c.426-1.756 2.924-1.756 3.35 0a1.724 1.724 0 002.573 1.066c1.543-.94 3.31.826 2.37 2.37a1.724 1.724 0 001.065 2.572c1.756.426 1.756 2.924 0 3.35a1.724 1.724 0 00-1.066 2.573c.94 1.543-.826 3.31-2.37 2.37a1.724 1.724 0 00-2.572 1.065c-.426 1.756-2.924 1.756-3.35 0a1.724 1.724 0 00-2.573-1.066c-1.543.94-3.31-.826-2.37-2.37a1.724 1.724 0 00-1.065-2.572c-1.756-.426-1.756-2.924 0-3.35a1.724 1.724 0 001.066-2.573c-.94-1.543.826-3.31 2.37-2.37a1.724 1.724 0 002.572-1.065z" /><path strokeLinecap="round" strokeLinejoin="round" d="M15 12a3 3 0 11-6 0 3 3 0 016 0z" /></svg>

</button>

<button onClick={() => setIsReportSectionOpen(!isReportSectionOpen)} className="px-4 py-2 rounded-md shadow-md transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.accentColor, color: currentTheme.textColor }}>Gerar Relatório</button>

<button onClick={handleAddClick} className="px-4 py-2 rounded-md shadow-md transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.primaryColor, color: currentTheme.textColor }}>+ Adicionar Atividade</button>

<button onClick={() => setShowGanttChart(!showGanttChart)} className="px-4 py-2 rounded-md shadow-md transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.secondaryColor, color: currentTheme.textColor }}>

{showGanttChart ? 'Ocultar Gantt' : 'Mostrar Gantt'}

</button>

<button onClick={() => setShowKanbanBoard(!showKanbanBoard)} className="px-4 py-2 rounded-md shadow-md transition-colors duration-200 text-sm" style={{ backgroundColor: currentTheme.warningColor, color: currentTheme.textColor }}>

{showKanbanBoard ? 'Ocultar Kanban' : 'Mostrar Kanban'}

</button>

</div>

</div>

{isTableSettingsOpen && (

<TableSettingsPanel

visibleColumns={visibleColumns} handleToggleColumnVisibility={handleToggleColumnVisibility}

feedbackMessage={feedbackMessage}

newStatusInput={newStatusInput} setNewStatusInput={setNewStatusInput}

newStatusColor={newStatusColor} setNewStatusColor={setNewStatusColor}

handleAddStatus={handleAddStatus} customStatuses={customStatuses}

newRiskInput={newRiskInput} setNewRiskInput={newRiskInput}

newRiskColor={newRiskColor} setNewRiskColor={setNewRiskColor}

handleAddRisk={handleAddRisk} customRisks={customRisks}

handleFileChange={handleFileChange} isImportingCsv={isImportingCsv}

handleExportCsv={handleExportCsv} csvFile={csvFile}

currentTheme={currentTheme}

getStatusTextColor={getStatusTextColor} getRiskTextColor={getRiskTextColor}

customColumns={customColumns} setCustomColumns={setCustomColumns}

allAvailableColumns={metrics.allAvailableColumns}

handleAddChartClick={handleAddChartClick}

showFeedback={showFeedback}

handleEditCustomColumn={handleEditCustomColumn}

columnError={columnError} setColumnError={setColumnError}

newColumnName={newColumnName} setNewColumnName={setNewColumnName} // Passed as props

newColumnType={newColumnType} setNewColumnType={setNewColumnType} // Passed as props

newColumnFormula={newColumnFormula} setNewColumnFormula={setNewColumnFormula} // Passed as props

/>

)}

<ActivityTableSection

activitiesWithCalculatedMetrics={metrics.activitiesWithCalculatedMetrics}

uniqueResponsaveis={metrics.uniqueResponsaveis}

customStatuses={customStatuses}

customRisks={customRisks}

customColumns={customColumns}

currentTheme={currentTheme}

searchTerm={searchTerm} setSearchTerm={setSearchTerm}

filterStatus={filterStatus} setFilterStatus={setFilterStatus}

filterResponsavel={filterResponsavel} setFilterResponsavel={setFilterResponsavel}

handleEditClick={handleEditClick}

handleDeleteClick={handleDeleteClick}

visibleColumns={visibleColumns}

getStatusBgColor={getStatusBgColor}

getStatusTextColor={getStatusTextColor}

getRiskBgColor={getRiskBgColor}

getRiskTextColor={getRiskTextColor}

/>

</section>

{showGanttChart && (

<GanttChart

activities={metrics.activitiesWithCalculatedMetrics}

currentTheme={currentTheme}

setAllActivities={setAllActivities}

showFeedback={showFeedback}

/>

)}

{showKanbanBoard && (

<KanbanBoard

activities={metrics.activitiesWithCalculatedMetrics}

onUpdateActivityStatus={handleUpdateActivityStatus}

customStatuses={customStatuses}

currentTheme={currentTheme}

getStatusBgColor={getStatusBgColor}

getStatusTextColor={getStatusTextColor}

/>

)}

{isReportSectionOpen && (

<ReportGeneratorSection

isReportSectionOpen={isReportSectionOpen}

setIsReportSectionOpen={setIsReportSectionOpen}

metrics={metrics}

customCharts={customCharts}

currentTheme={currentTheme}

isGeneratingPdf={isGeneratingPdf}

pdfLibsLoaded={pdfLibsLoaded}

handleExportPdf={handleExportPdf}

startDate={startDate}

endDate={endDate}

getStatusBgColor={getStatusBgColor}

getStatusTextColor={getStatusTextColor}

/>

)}

{isEditModalOpen && (

<EditActivityModal

activity={currentActivityToEdit}

onClose={handleCloseEditModal}

onSave={handleSaveActivity}

customStatuses={customStatuses}

customRisks={customRisks}

customColumns={customColumns}

currentTheme={currentTheme}

/>

)}

{isAddModalOpen && (

<AddActivityModal

onClose={handleCloseAddModal}

onSave={handleAddNewActivity}

customStatuses={customStatuses}

customRisks={customRisks}

customColumns={customColumns}

currentTheme={currentTheme}

/>

)}

{showCsvPreviewModal && (

<CsvPreviewModal

headers={csvPreviewHeaders}

data={csvPreviewData}

onClose={handleCancelImport}

onConfirm={handleConfirmImport}

currentTheme={currentTheme}

/>

)}

{isCreateChartModalOpen && (

<CreateChartModal

onClose={handleCloseCreateChartModal}

onSave={handleSaveCustomChart}

allAvailableColumns={metrics.allAvailableColumns}

currentTheme={currentTheme}

/>

)}

{isDeleteConfirmModalOpen && (

<DeleteConfirmationModal

itemToDelete={itemToDelete}

itemType={itemTypeToDelete}

onClose={handleCancelDelete}

onConfirm={handleConfirmDelete}

currentTheme={currentTheme}

/>

)}

{isEditCustomColumnModalOpen && (

<EditCustomColumnModal

column={currentCustomColumnToEdit}

onClose={handleCloseEditCustomColumnModal}

onSave={handleSaveEditedCustomColumn}

currentTheme={currentTheme}

/>

)}

<script src="https://cdn.tailwindcss.com"></script>

<style>

{`

@import url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap');

body { font-family: 'Inter', sans-serif; }

@media print {

body > \*:not(#report-content) { display: none; }

#report-content {

display: block; width: 100%; margin: 0; padding: 0; box-shadow: none; border-radius: 0;

}

.no-print { display: none !important; }

table { width: 100%; border-collapse: collapse; }

th, td { border: 1px solid #ddd; padding: 8px; font-size: 0.75em; }

.recharts-responsive-container { width: 100% !important; height: 200px !important; margin-bottom: 10px; }

.px-2.inline-flex.text-xs.leading-5.font-semibold.rounded-full { -webkit-print-color-adjust: exact; color-adjust: exact; }

h2 { page-break-before: always; }

h3 { page-break-after: avoid; }

.grid { display: block; }

.grid > div { width: 100%; margin-bottom: 10px; }

}

`}

</style>

</div>

);

};

export default App;