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

import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';

import './App.css';

import './index.css';

// Componentes

import LoginForm from './components/LoginForm';

import ProtectedRoute from './components/ProtectedRoute';

import Sidebar from './components/Sidebar';

import ConsolidatedDashboard from './components/ConsolidatedDashboard';

import IntegratedPMODashboard from './components/IntegratedPMODashboard';

import PortfolioStrategic from './components/PortfolioStrategic';

import ProjectManagementTabs from './components/ProjectManagementTabs';

import ScheduleManagement from './components/ScheduleManagement';

import FinancialManagement from './components/FinancialManagement';

import ResourceManagement from './components/ResourceManagement';

import RiskManagement from './components/RiskManagement';

import ChangeManagement from './components/ChangeManagement';

import CashFlowProjection from './components/CashFlowProjection';

import FileManager from './components/FileManager';

import ReportsManagement from './components/ReportsManagement';

import ProjectAudit from './components/ProjectAudit';

import ProjectArchive from './components/ProjectArchive';

import SyncIndicator from './components/SyncIndicator';

import BackupManager from './components/BackupManager';

import FileStatusIndicator from './components/FileStatusIndicator';

import SupabaseAuth from './components/SupabaseAuth';

import OrganizationMembers from './components/OrganizationMembers';

import UserManagement from './components/UserManagement';

import SplashScreen from './components/SplashScreen';

import filePersistenceService from './services/FilePersistenceService';

import supabaseService from './services/SupabaseService';

// Contextos

import { AuthProvider, useAuth } from './contexts/AuthContext';

import { ProjectProvider } from './contexts/ProjectContext';

// Componente principal de la aplicaciÃ³n (requiere autenticaciÃ³n)

function MainApp() {

const { user, showSplash, completeSplash } = useAuth();

// ===== ESTADO MULTI-PROYECTO =====

const [projects, setProjects] = useState([

// Proyecto de demostraciÃ³n para familiarizarse con el sistema

{

id: 'demo-001',

name: 'Proyecto de DemostraciÃ³n',

description: 'Proyecto de ejemplo para familiarizarse con el sistema. Puede eliminarse cuando agregue sus proyectos reales.',

status: 'active',

priority: 'medium',

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

endDate: new Date(Date.now() + 30 \* 24 \* 60 \* 60 \* 1000).toISOString().split('T')[0], // 30 dÃ­as desde hoy

budget: 100000,

contingencyReserve: 10000,

managementReserve: 5000,

manager: 'Usuario Demo',

sponsor: 'OrganizaciÃ³n',

createdAt: new Date().toISOString(),

updatedAt: new Date().toISOString(),

version: '1.0.0',

progress: 0

}

]);

// FunciÃ³n para obtener el proyecto inicial con lÃ³gica de respaldo

const getInitialProjectId = () => {

// 1. Intentar cargar el Ãºltimo proyecto seleccionado desde localStorage

const lastSelectedProjectId = localStorage.getItem('lastSelectedProjectId');

if (lastSelectedProjectId) {

// Verificar si el proyecto aÃºn existe y estÃ¡ activo

const project = projects.find(p => p.id === lastSelectedProjectId);

if (project && project.status === 'active') {

return lastSelectedProjectId;

}

}

// 2. Si no hay proyecto seleccionado o no estÃ¡ activo, buscar el primer proyecto activo

const activeProject = projects.find(p => p.status === 'active');

if (activeProject) {

return activeProject.id;

}

// 3. Si no hay proyectos activos, usar el primer proyecto disponible

if (projects.length > 0) {

return projects[0].id;

}

// 4. Fallback por defecto

return 'demo-001';

};

const [currentProjectId, setCurrentProjectId] = useState(getInitialProjectId);

const [portfolioViewMode, setPortfolioViewMode] = useState('portfolio'); // 'portfolio', 'project', 'schedule'

const [viewMode, setViewMode] = useState('dashboard');

// Estado para controlar el modo de persistencia

const [useSupabase, setUseSupabase] = useState(false);

const [supabaseInitialized, setSupabaseInitialized] = useState(false);

const [showAuthModal, setShowAuthModal] = useState(false);

// Corregir currentProjectId si no coincide con ningÃºn proyecto existente

useEffect(() => {

if (projects && projects.length > 0) {

const projectExists = projects.find(p => p.id === currentProjectId);

if (!projectExists) {

// Buscar el primer proyecto activo disponible

const activeProject = projects.find(p => p.status === 'active');

const fallbackProjectId = activeProject ? activeProject.id : projects[0].id;

console.log('ðŸ”§ CORRIGIENDO currentProjectId:', {

currentProjectId,

availableProjects: projects.map(p => ({ id: p.id, name: p.name, status: p.status })),

fallbackTo: fallbackProjectId,

reason: 'Proyecto no encontrado, usando primer proyecto disponible'

});

setCurrentProjectId(fallbackProjectId);

}

}

}, [projects, currentProjectId]);

// Work Packages eliminados - ya no se usan

// Tareas por proyecto - cada proyecto tiene su propio cronograma

const [tasksByProject, setTasksByProject] = useState({

'demo-001': [] // Proyecto de demostraciÃ³n - sin tareas iniciales

});

// ConfiguraciÃ³n de dÃ­as laborables por proyecto - cada proyecto puede tener su propia configuraciÃ³n

const [includeWeekendsByProject, setIncludeWeekendsByProject] = useState({

'demo-001': false // Por defecto: solo dÃ­as laborales (lunes a viernes)

});

// FunciÃ³n para limpiar datos corruptos - CON MONITOREO INTELIGENTE

const cleanTasksByProject = (data) => {

if (!data || typeof data !== 'object') {

console.warn('ðŸš¨ CORRUPCIÃ“N REAL: data no es un objeto vÃ¡lido');

return {};

}

const cleaned = {};

let hasRealCorruption = false;

let corruptionDetails = [];

Object.keys(data).forEach(projectId => {

const projectTasks = data[projectId];

// Detectar corrupciÃ³n REAL (no arrays vÃ¡lidos)

if (typeof projectTasks === 'function') {

console.warn(`ðŸš¨ CORRUPCIÃ“N REAL detectada en proyecto ${projectId}: funciÃ³n en lugar de array`);

corruptionDetails.push(`${projectId}: funciÃ³n`);

cleaned[projectId] = [];

hasRealCorruption = true;

} else if (projectTasks && !Array.isArray(projectTasks) && typeof projectTasks === 'object' && projectTasks.constructor !== Array) {

console.warn(`ðŸš¨ CORRUPCIÃ“N REAL detectada en proyecto ${projectId}: objeto no-array`);

corruptionDetails.push(`${projectId}: objeto no-array`);

cleaned[projectId] = [];

hasRealCorruption = true;

} else {

// Datos vÃ¡lidos - mantener tal como estÃ¡n

cleaned[projectId] = projectTasks;

}

});