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

import {

Search, Filter, Calendar, Users, CheckCircle, Clock,

AlertCircle, ChevronDown, ChevronRight, X, Plus,

Download, RefreshCw, Eye, Trash2, Edit, MoreVertical,

Star, Save, FolderOpen, ChevronUp, Paperclip,

UserPlus, Building, Phone, MessageSquare, Lock,

Repeat, FileText, Columns, ArrowUpDown, Loader,

Check, Info, Edit2, GripVertical, User

} from 'lucide-react';

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

SEZIONE 1: COMPONENTI CONDIVISI (UTILITY)

- SearchableSelect: Dropdown con ricerca

- Toast: Notifiche

- ExpandableSection: Sezioni espandibili

- ToggleSwitch: Switch moderni (NUOVO)

- InlineDropdown: Dropdown per modifica inline (NUOVO)

- InlinePriorityEdit: Modifica priorità inline (NUOVO)

- InlineAssigneeEdit: Modifica assegnatario inline (NUOVO)

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

// 1.1 SearchableSelect - Dropdown con ricerca real-time

const SearchableSelect = ({

label,

placeholder,

value,

onChange,

options = [],

icon: Icon,

onSearch,

multiple = false,

required = false,

className = ""

}) => {

const [isOpen, setIsOpen] = useState(false);

const [searchTerm, setSearchTerm] = useState('');

const [searchResults, setSearchResults] = useState(options);

const [isSearching, setIsSearching] = useState(false);

const dropdownRef = useRef(null);

// Chiudi dropdown quando clicchi fuori

useEffect(() => {

const handleClickOutside = (event) => {

if (dropdownRef.current && !dropdownRef.current.contains(event.target)) {

setIsOpen(false);

}

};

document.addEventListener('mousedown', handleClickOutside);

return () => document.removeEventListener('mousedown', handleClickOutside);

}, []);

// Gestione ricerca con debouncing

useEffect(() => {

const delayDebounceFn = setTimeout(async () => {

if (searchTerm) {

setIsSearching(true);

if (onSearch) {

const results = await onSearch(searchTerm);

setSearchResults(results);

} else {

const filtered = options.filter(opt =>

opt.label.toLowerCase().includes(searchTerm.toLowerCase())

);

setSearchResults(filtered);

}

setIsSearching(false);

} else {

setSearchResults(options);

}

}, 300);

return () => clearTimeout(delayDebounceFn);

}, [searchTerm, options, onSearch]);

const selectedOption = options.find(opt => opt.value === value);

return (

<div className={`relative ${className}`} ref={dropdownRef}>

{label && (

<label className="block text-sm font-medium mb-1">

{label} {required && <span className="text-red-500">\*</span>}

</label>

)}

<div

className="relative cursor-pointer"

onClick={() => setIsOpen(!isOpen)}

>

<input

type="text"

placeholder={placeholder}

value={selectedOption?.label || ''}

readOnly

className="w-full px-3 py-2 pr-10 border rounded-lg cursor-pointer hover:border-gray-400 focus:ring-2 focus:ring-blue-500 focus:border-transparent"

/>

{Icon && !label && (

<Icon className="absolute left-3 top-1/2 -translate-y-1/2 w-4 h-4 text-gray-400" />

)}

<ChevronDown className={`absolute right-3 top-1/2 -translate-y-1/2 w-4 h-4 text-gray-400 transition-transform ${isOpen ? 'rotate-180' : ''}`} />

</div>

{isOpen && (

<div className="absolute z-50 w-full mt-1 bg-white border rounded-lg shadow-lg max-h-60 overflow-hidden">

<div className="p-2 border-b sticky top-0 bg-white">

<div className="relative">

<Search className="absolute left-3 top-1/2 -translate-y-1/2 w-4 h-4 text-gray-400" />

<input

type="text"

placeholder={`Cerca${label ? ' ' + label.toLowerCase() : ''}...`}

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

className="w-full pl-10 pr-3 py-2 border rounded-md focus:ring-2 focus:ring-blue-500 focus:border-transparent"

onClick={(e) => e.stopPropagation()}

/>

{isSearching && (

<Loader className="absolute right-3 top-1/2 -translate-y-1/2 w-4 h-4 animate-spin text-gray-400" />

)}

</div>

</div>

<div className="max-h-48 overflow-y-auto">

{searchResults.length > 0 ? (

searchResults.map(option => (

<div

key={option.value}

className="px-3 py-2 hover:bg-gray-100 cursor-pointer flex items-center justify-between"

onClick={() => {

onChange(option.value);

setIsOpen(false);

setSearchTerm('');

}}

>

<span className="text-sm">{option.label}</span>

{value === option.value && <Check className="w-4 h-4 text-blue-600" />}

</div>

))

) : (

<div className="px-3 py-8 text-center text-gray-500 text-sm">

Nessun risultato trovato

</div>

)}

</div>

</div>

)}

</div>

);

};

// 1.2 Toast - Componente notifiche

const Toast = ({ message, type = 'success', onClose }) => {

useEffect(() => {

const timer = setTimeout(onClose, 3000);

return () => clearTimeout(timer);

}, [onClose]);

const bgColor = type === 'success' ? 'bg-green-500' : type === 'error' ? 'bg-red-500' : 'bg-blue-500';

const icon = type === 'success' ? <Check className="w-5 h-5" /> :

type === 'error' ? <X className="w-5 h-5" /> :

<Info className="w-5 h-5" />;

return (

<div className={`fixed bottom-4 right-4 ${bgColor} text-white px-6 py-3 rounded-lg shadow-lg flex items-center gap-3 animate-slideUp z-50`}>

{icon}

<span>{message}</span>

<button onClick={onClose} className="hover:opacity-80">

<X className="w-4 h-4" />

</button>

</div>

);

};

// 1.3 ExpandableSection - Sezioni espandibili con chevron

const ExpandableSection = ({ title, children, defaultOpen = false, icon: Icon }) => {

const [isOpen, setIsOpen] = useState(defaultOpen);

return (

<div className="border-t pt-4 mt-4">

<button

onClick={() => setIsOpen(!isOpen)}

className="flex items-center gap-2 w-full text-left hover:text-gray-700 transition-colors"

>

<ChevronRight className={`w-4 h-4 transition-transform ${isOpen ? 'rotate-90' : ''}`} />

{Icon && <Icon className="w-4 h-4 text-gray-500" />}

<span className="font-medium">{title}</span>

</button>

{isOpen && (

<div className="mt-4 pl-6 animate-slideDown">

{children}

</div>

)}

</div>

);

};

// 1.4 ToggleSwitch - Switch toggle moderno con animazioni (NUOVO)

const ToggleSwitch = ({

checked = false,

onChange,

label,

disabled = false,

size = 'default',

showIcons = false,

variant = 'default'

}) => {

const sizes = {

small: {

switch: 'w-8 h-4',

ball: 'w-3 h-3',

translate: 'translate-x-4',

icon: 'w-2 h-2'

},

default: {

switch: 'w-11 h-6',

ball: 'w-5 h-5',

translate: 'translate-x-5',

icon: 'w-3 h-3'

},

large: {

switch: 'w-14 h-8',

ball: 'w-6 h-6',

translate: 'translate-x-6',

icon: 'w-4 h-4'

}

};

const variants = {

default: {

checked: 'bg-blue-600',

unchecked: 'bg-gray-300',

focus: 'focus:ring-blue-500'

},

success: {

checked: 'bg-green-600',

unchecked: 'bg-gray-300',

focus: 'focus:ring-green-500'

},

danger: {

checked: 'bg-red-600',

unchecked: 'bg-gray-300',

focus: 'focus:ring-red-500'

},

ios: {

checked: 'bg-green-500',

unchecked: 'bg-gray-400',

focus: 'focus:ring-green-400'

}

};

const currentSize = sizes[size];

const currentVariant = variants[variant];

return (

<label className={`flex items-center gap-3 cursor-pointer ${disabled ? 'opacity-50 cursor-not-allowed' : ''}`}>

<div className="relative">

<input

type="checkbox"

className="sr-only"

checked={checked}

onChange={onChange}

disabled={disabled}

/>

<div

className={`

block ${currentSize.switch} rounded-full transition-colors duration-200

${checked ? currentVariant.checked : currentVariant.unchecked}

${disabled ? '' : 'hover:shadow-md'}

`}

/>

<div

className={`

absolute top-0.5 left-0.5 ${currentSize.ball}

bg-white rounded-full shadow transition-transform duration-200

flex items-center justify-center

${checked ? currentSize.translate : 'translate-x-0'}

`}

>

{showIcons && (

checked ?

<Check className={`${currentSize.icon} text-green-600`} /> :

<X className={`${currentSize.icon} text-gray-400`} />

)}

</div>

</div>

{label && (

<span className={`text-sm select-none ${disabled ? 'text-gray-400' : ''}`}>

{label}

</span>

)}

</label>

);

};

// 1.5 InlineDropdown - Dropdown modificabile inline generico (NUOVO)

const InlineDropdown = ({

value,

options = [],

onChange,

onCancel,

renderValue,

renderOption,

searchable = false,

placeholder = "Seleziona...",

loading = false

}) => {

const [isOpen, setIsOpen] = useState(true);

const [searchTerm, setSearchTerm] = useState('');

const dropdownRef = useRef(null);

useEffect(() => {

const handleClickOutside = (event) => {

if (dropdownRef.current && !dropdownRef.current.contains(event.target)) {

onCancel();

}

};

document.addEventListener('mousedown', handleClickOutside);

return () => document.removeEventListener('mousedown', handleClickOutside);

}, [onCancel]);

const filteredOptions = searchable && searchTerm

? options.filter(opt =>

opt.label.toLowerCase().includes(searchTerm.toLowerCase())

)

: options;

return (

<div ref={dropdownRef} className="relative">

<div className="absolute z-50 mt-1 w-48 bg-white rounded-lg shadow-lg border">

{searchable && (

<div className="p-2 border-b">

<div className="relative">

<Search className="absolute left-2 top-1/2 -translate-y-1/2 w-4 h-4 text-gray-400" />

<input

type="text"

placeholder="Cerca..."

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

className="w-full pl-8 pr-2 py-1 text-sm border rounded focus:ring-2 focus:ring-blue-500"

autoFocus

/>

</div>

</div>

)}

<div className="max-h-60 overflow-y-auto py-1">

{loading ? (

<div className="px-3 py-4 text-center">

<Loader className="w-5 h-5 animate-spin mx-auto text-gray-400" />

</div>

) : filteredOptions.length === 0 ? (

<div className="px-3 py-4 text-center text-sm text-gray-500">

Nessun risultato

</div>

) : (

filteredOptions.map((option) => (

<button

key={option.value}

onClick={() => onChange(option)}

className="w-full px-3 py-2 text-left hover:bg-gray-100 flex items-center justify-between group"

>

{renderOption ? renderOption(option) : (

<span className="text-sm">{option.label}</span>

)}

{value === option.value && (

<Check className="w-4 h-4 text-blue-600" />

)}

</button>

))

)}

</div>

<div className="border-t p-2 flex justify-end gap-2">

<button

onClick={onCancel}

className="px-3 py-1 text-sm text-gray-600 hover:text-gray-800"

>

Annulla

</button>

</div>

</div>

</div>

);

};

// 1.6 InlinePriorityEdit - Modifica priorità inline (NUOVO)

const InlinePriorityEdit = ({

priority,

onSave,

onCancel

}) => {

const priorityOptions = [

{

value: 'low',

label: 'Bassa',

color: 'bg-green-100 text-green-700',

icon: '🟢'

},

{

value: 'normal',

label: 'Normale',

color: 'bg-blue-100 text-blue-700',

icon: '🔵'

},

{

value: 'high',

label: 'Alta',

color: 'bg-orange-100 text-orange-700',

icon: '🟠'

},

{

value: 'urgent',

label: 'Urgente',

color: 'bg-red-100 text-red-700',

icon: '🔴'

}

];

return (

<InlineDropdown

value={priority}

options={priorityOptions}

onChange={(option) => onSave(option.value)}

onCancel={onCancel}

renderOption={(option) => (

<div className="flex items-center gap-2">

<span>{option.icon}</span>

<span className={`px-2 py-0.5 rounded-full text-xs font-medium ${option.color}`}>

{option.label}

</span>

</div>

)}

/>

);

};

// 1.7 InlineAssigneeEdit - Modifica assegnatario inline (NUOVO)

const InlineAssigneeEdit = ({

assigneeId,

assignees = [],

onSave,

onCancel,

onSearch

}) => {

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

const [options, setOptions] = useState(assignees);

const handleSearch = async (term) => {

if (onSearch) {

setLoading(true);

const results = await onSearch(term);

setOptions(results);

setLoading(false);

}

};

return (

<InlineDropdown

value={assigneeId}

options={options}

onChange={(option) => onSave(option.value)}

onCancel={onCancel}

searchable={true}

loading={loading}

renderOption={(option) => (

<div className="flex items-center gap-2">

<div className="w-6 h-6 bg-blue-500 text-white rounded-full flex items-center justify-center text-xs">

{option.label.substring(0, 2).toUpperCase()}

</div>

<span className="text-sm">{option.label}</span>

</div>

)}

/>

);

};

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

SEZIONE 2: QUICK ADD TASK (Componente B)

- Form rapido (input singolo)

- Form espanso completo

- Gestione sottoattività

- Upload documenti

- ToggleSwitch integrati (MODIFICATO)

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

// 2.1 SubtaskList - Gestione sottoattività

const SubtaskList = ({ subtasks = [], onChange }) => {

const [newSubtask, setNewSubtask] = useState('');

const addSubtask = () => {

if (newSubtask.trim()) {

onChange([...subtasks, {

id: Date.now(),

title: newSubtask,

completed: false

}]);

setNewSubtask('');

}

};

return (

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

<div className="flex gap-2">

<input

type="text"

placeholder="Aggiungi sottoattività"

value={newSubtask}

onChange={(e) => setNewSubtask(e.target.value)}

onKeyPress={(e) => e.key === 'Enter' && addSubtask()}

className="flex-1 px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

/>

<button className="p-2 hover:bg-gray-100 rounded" title="Calendario">

<Calendar className="w-5 h-5 text-gray-600" />

</button>

<button className="p-2 hover:bg-gray-100 rounded" title="Assegnatari">

<Users className="w-5 h-5 text-gray-600" />

</button>

<button className="p-2 hover:bg-gray-100 rounded" title="Telefono">

<Phone className="w-5 h-5 text-gray-600" />

</button>

</div>

{subtasks.map((subtask, index) => (

<div key={subtask.id} className="flex items-center gap-2 p-2 hover:bg-gray-50 rounded">

<input

type="checkbox"

checked={subtask.completed}

onChange={() => {

const updated = [...subtasks];

updated[index].completed = !updated[index].completed;

onChange(updated);

}}

className="rounded text-blue-600"

/>

<span className={`flex-1 ${subtask.completed ? 'line-through text-gray-500' : ''}`}>

{subtask.title}

</span>

<button

onClick={() => onChange(subtasks.filter((\_, i) => i !== index))}

className="text-red-500 hover:text-red-700 opacity-0 hover:opacity-100 transition-opacity"

>

<X className="w-4 h-4" />

</button>

</div>

))}

</div>

);

};

// 2.2 QuickAddTask - Componente principale per aggiungere task (MODIFICATO CON TOGGLESWITCH)

const QuickAddTask = ({ onAdd, clients = [], contacts = [], assignees = [] }) => {

const [isExpanded, setIsExpanded] = useState(false);

const [quickTitle, setQuickTitle] = useState('');

const [isSubmitting, setIsSubmitting] = useState(false);

const quickInputRef = useRef(null);

// 2.2.1 Form data state

const [formData, setFormData] = useState({

title: '',

description: '',

priority: 'normal',

assignees: [],

due\_date: new Date().toISOString().split('T')[0],

due\_time: '23:59',

observer\_id: null,

client\_id: null,

contact\_id: null,

feedback\_requested: false,

is\_private: false,

repeat\_task: false,

documents: [],

subtasks: []

});

// 2.2.2 Reset form quando chiudi

useEffect(() => {

if (!isExpanded) {

setFormData({

title: '',

description: '',

priority: 'normal',

assignees: [],

due\_date: new Date().toISOString().split('T')[0],

due\_time: '23:59',

observer\_id: null,

client\_id: null,

contact\_id: null,

feedback\_requested: false,

is\_private: false,

repeat\_task: false,

documents: [],

subtasks: []

});

}

}, [isExpanded]);

// 2.2.3 Handler aggiunta rapida

const handleQuickAdd = async () => {

if (quickTitle.trim() && !isSubmitting) {

setIsSubmitting(true);

try {

await onAdd({

title: quickTitle,

assignee\_id: 'current\_user',

status: 'pending',

priority: 'normal'

});

setQuickTitle('');

quickInputRef.current?.focus();

} catch (error) {

console.error('Errore:', error);

} finally {

setIsSubmitting(false);

}

}

};

// 2.2.4 Handler submit form completo

const handleSubmit = async () => {

if (formData.title.trim() && !isSubmitting) {

setIsSubmitting(true);

try {

await onAdd(formData);

setIsExpanded(false);

} catch (error) {

console.error('Errore:', error);

} finally {

setIsSubmitting(false);

}

}

};

// 2.2.5 Funzioni ricerca asincrona

const searchClients = async (term) => {

await new Promise(resolve => setTimeout(resolve, 300));

return clients.filter(c => c.label.toLowerCase().includes(term.toLowerCase()));

};

const searchContacts = async (term) => {

await new Promise(resolve => setTimeout(resolve, 300));

return contacts.filter(c => c.label.toLowerCase().includes(term.toLowerCase()));

};

// 2.2.6 Render modalità rapida

if (!isExpanded) {

return (

<div className="flex gap-2">

<input

ref={quickInputRef}

type="text"

placeholder="Nome attività"

value={quickTitle}

onChange={(e) => setQuickTitle(e.target.value)}

onKeyPress={(e) => {

if (e.key === 'Enter') {

e.preventDefault();

handleQuickAdd();

}

}}

disabled={isSubmitting}

className="flex-1 px-4 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500 focus:border-transparent transition-all"

/>

<button

onClick={() => setIsExpanded(true)}

className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors"

title="Aggiungi attività dettagliata"

>

<Plus className="w-5 h-5" />

</button>

</div>

);

}

// 2.2.7 Render modalità espansa

return (

<div className="bg-white rounded-lg shadow-lg border p-6 animate-fadeIn">

<h3 className="text-lg font-semibold mb-4">Nuova Attività</h3>

{/\* Titolo del lavoro \*/}

<div className="mb-4">

<label className="block text-sm font-medium mb-1">Titolo del lavoro</label>

<div className="flex items-center gap-2">

<CheckCircle className="w-5 h-5 text-gray-400" />

<input

type="text"

placeholder="ad es. Attività da fare"

value={formData.title}

onChange={(e) => setFormData({...formData, title: e.target.value})}

className="flex-1 px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

autoFocus

/>

<button className="p-2 hover:bg-gray-100 rounded" title="Calendario">

<Calendar className="w-5 h-5 text-gray-600" />

</button>

<button className="p-2 hover:bg-gray-100 rounded" title="Assegnatari">

<Users className="w-5 h-5 text-gray-600" />

</button>

<button className="p-2 hover:bg-gray-100 rounded" title="Telefono">

<Phone className="w-5 h-5 text-gray-600" />

</button>

<span className="text-gray-400">0)</span>

</div>

</div>

{/\* Campo Priorità \*/}

<div className="mb-4">

<label className="block text-sm font-medium mb-1">Priorità</label>

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

{[

{ value: 'low', label: 'Bassa', color: 'bg-green-100 text-green-700 border-green-300' },

{ value: 'normal', label: 'Normale', color: 'bg-blue-100 text-blue-700 border-blue-300' },

{ value: 'high', label: 'Alta', color: 'bg-orange-100 text-orange-700 border-orange-300' },

{ value: 'urgent', label: 'Urgente', color: 'bg-red-100 text-red-700 border-red-300' }

].map((priority) => (

<button

key={priority.value}

type="button"

onClick={() => setFormData({...formData, priority: priority.value})}

className={`px-3 py-2 rounded-lg border-2 font-medium transition-all ${

formData.priority === priority.value

? priority.color + ' border-opacity-100'

: 'bg-gray-50 text-gray-600 border-gray-200 hover:bg-gray-100'

}`}

>

{priority.label}

</button>

))}

</div>

</div>

{/\* Descrizione \*/}

<div className="mb-4">

<label className="block text-sm font-medium mb-1">Descrizione</label>

<textarea

placeholder="Descrizione..."

value={formData.description}

onChange={(e) => setFormData({...formData, description: e.target.value})}

className="w-full px-3 py-2 border rounded-lg resize-none focus:ring-2 focus:ring-blue-500"

rows={3}

/>

</div>

{/\* Assegnatari \*/}

<div className="mb-4">

<SearchableSelect

label="Assegnatari"

placeholder="Seleziona assegnatari..."

value={formData.assignees[0]}

onChange={(value) => setFormData({...formData, assignees: [value]})}

options={assignees}

icon={Users}

/>

</div>

{/\* Data e Ora \*/}

<div className="grid grid-cols-2 gap-4 mb-4">

<div>

<label className="block text-sm font-medium mb-1">

<Calendar className="w-4 h-4 inline mr-1" />

Data

</label>

<input

type="date"

value={formData.due\_date}

onChange={(e) => setFormData({...formData, due\_date: e.target.value})}

className="w-full px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium mb-1">

<Clock className="w-4 h-4 inline mr-1" />

Ora

</label>

<input

type="time"

value={formData.due\_time}

onChange={(e) => setFormData({...formData, due\_time: e.target.value})}

className="w-full px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

{/\* Sezioni espandibili \*/}

<ExpandableSection

title="Altro (Cliente, Contatto, Richiesta feedback, Attività privata, Ripeti attività)"

icon={ChevronDown}

>

<div className="space-y-4">

<SearchableSelect

label="Osservatore"

placeholder="Seleziona osservatore..."

value={formData.observer\_id}

onChange={(value) => setFormData({...formData, observer\_id: value})}

options={assignees}

icon={Eye}

/>

<SearchableSelect

label="Cliente"

placeholder="Seleziona cliente..."

value={formData.client\_id}

onChange={(value) => setFormData({...formData, client\_id: value})}

options={clients}

onSearch={searchClients}

icon={Building}

/>

<SearchableSelect

label="Contatto"

placeholder="Seleziona contatto..."

value={formData.contact\_id}

onChange={(value) => setFormData({...formData, contact\_id: value})}

options={contacts}

onSearch={searchContacts}

icon={UserPlus}

/>

{/\* MODIFICATO: Uso ToggleSwitch invece di checkbox \*/}

<div className="space-y-3 pt-2">

<ToggleSwitch

checked={formData.feedback\_requested}

onChange={(e) => setFormData({...formData, feedback\_requested: e.target.checked})}

label="Feedback Richiesto"

variant="default"

/>

<ToggleSwitch

checked={formData.is\_private}

onChange={(e) => setFormData({...formData, is\_private: e.target.checked})}

label="Attività Privata"

variant="danger"

showIcons={true}

/>

<ToggleSwitch

checked={formData.repeat\_task}

onChange={(e) => setFormData({...formData, repeat\_task: e.target.checked})}

label="Ripeti Attività"

variant="success"

/>

</div>

</div>

</ExpandableSection>

<ExpandableSection title="Documenti allegati" icon={Paperclip} defaultOpen={false}>

<div

className="border-2 border-dashed rounded-lg p-4 transition-colors"

onDragOver={(e) => {

e.preventDefault();

e.currentTarget.classList.add('border-blue-500', 'bg-blue-50');

}}

onDragLeave={(e) => {

e.currentTarget.classList.remove('border-blue-500', 'bg-blue-50');

}}

onDrop={(e) => {

e.preventDefault();

e.currentTarget.classList.remove('border-blue-500', 'bg-blue-50');

const files = Array.from(e.dataTransfer.files);

setFormData({...formData, documents: [...formData.documents, ...files]});

}}

>

<input

type="file"

multiple

className="hidden"

id="file-upload"

onChange={(e) => {

const files = Array.from(e.target.files);

setFormData({...formData, documents: [...formData.documents, ...files]});

}}

/>

<label htmlFor="file-upload" className="cursor-pointer">

<div className="text-center">

<Paperclip className="w-8 h-8 mx-auto mb-2 text-gray-400" />

<p className="text-sm text-gray-600 uppercase font-medium">Trascina file qui</p>

<p className="text-xs text-gray-500 mt-1">oppure clicca per selezionare</p>

</div>

</label>

{formData.documents.length > 0 && (

<div className="mt-4 space-y-2">

{formData.documents.map((file, index) => {

const isImage = file.type.startsWith('image/');

const isPDF = file.type === 'application/pdf';

return (

<div key={index} className="flex items-center gap-3 p-2 bg-gray-50 rounded-lg">

{/\* File icon/preview \*/}

<div className="w-10 h-10 flex items-center justify-center bg-white rounded border">

{isImage ? (

<img

src={URL.createObjectURL(file)}

alt={file.name}

className="w-full h-full object-cover rounded"

/>

) : isPDF ? (

<FileText className="w-5 h-5 text-red-500" />

) : (

<Paperclip className="w-5 h-5 text-gray-500" />

)}

</div>

{/\* File info \*/}

<div className="flex-1 min-w-0">

<p className="text-sm font-medium truncate">{file.name}</p>

<p className="text-xs text-gray-500">

{(file.size / 1024 / 1024).toFixed(2)} MB

</p>

</div>

{/\* Remove button \*/}

<button

onClick={() => {

const newDocs = formData.documents.filter((\_, i) => i !== index);

setFormData({...formData, documents: newDocs});

}}

className="p-1 text-red-500 hover:text-red-700 hover:bg-red-50 rounded"

>

<X className="w-4 h-4" />

</button>

</div>

);

})}

</div>

)}

</div>

</ExpandableSection>

<ExpandableSection title="Sottoattività" icon={CheckCircle} defaultOpen={false}>

<SubtaskList

subtasks={formData.subtasks}

onChange={(subtasks) => setFormData({...formData, subtasks})}

/>

</ExpandableSection>

{/\* Pulsanti azione \*/}

<div className="flex justify-between mt-6 pt-4">

<button

onClick={() => setIsExpanded(false)}

disabled={isSubmitting}

className="px-6 py-2 bg-red-500 text-white rounded-lg hover:bg-red-600 transition-colors disabled:opacity-50"

>

Annulla

</button>

<button

onClick={handleSubmit}

disabled={!formData.title.trim() || isSubmitting}

className="px-6 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:opacity-50 disabled:cursor-not-allowed transition-all flex items-center gap-2"

>

{isSubmitting ? (

<>

<Loader className="w-4 h-4 animate-spin" />

Salvataggio...

</>

) : (

'Aggiungi'

)}

</button>

</div>

</div>

);

};

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

SEZIONE 3: TASK TABLE (Componente A)

- Tabella principale con Drag & Drop (NUOVO)

- Filtri rapidi

- Ricerca

- Azioni bulk

- Ordinamento

- Modifica inline avanzata (NUOVO)

- Highlight righe (NUOVO)

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

// 3.1 TaskSkeleton - Loading placeholder

const TaskSkeleton = () => (

<tr className="animate-pulse">

<td className="p-2 w-8"><div className="w-4 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-5 h-5 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-5 h-5 bg-gray-200 rounded-full"></div></td>

<td className="p-3"><div className="w-40 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-20 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-60 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-24 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-32 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-8 h-8 bg-gray-200 rounded-full"></div></td>

<td className="p-3"><div className="w-32 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-20 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-20 h-4 bg-gray-200 rounded"></div></td>

<td className="p-3"><div className="w-4 h-4 bg-gray-200 rounded"></div></td>

</tr>

);

// 3.2 TaskRow - Singola riga con modifica inline avanzata (MODIFICATO)

const TaskRow = ({

task,

isSelected,

onToggleSelect,

onEdit,

onDelete,

onStatusChange,

searchTerm = '',

onInlineUpdate,

assignees = [],

isDragging,

dragHandleProps

}) => {

const [showActions, setShowActions] = useState(false);

const [editingField, setEditingField] = useState(null);

const [editValues, setEditValues] = useState({

title: task.title,

description: task.description || '',

due\_date: task.due\_date

});

const inputRef = useRef(null);

useEffect(() => {

if (editingField && inputRef.current) {

inputRef.current.focus();

if (inputRef.current.select) {

inputRef.current.select();

}

}

}, [editingField]);

const formatDate = (date) => {

if (!date) return '-';

const d = new Date(date);

return d.toLocaleDateString('it-IT', { day: '2-digit', month: '2-digit', year: 'numeric' });

};

const highlightText = (text, search) => {

if (!search || !text) return text;

const parts = text.split(new RegExp(`(${search})`, 'gi'));

return parts.map((part, index) =>

part.toLowerCase() === search.toLowerCase() ? (

<mark key={index} className="bg-yellow-200 px-0.5 rounded">

{part}

</mark>

) : (

part

)

);

};

const isNewTask = () => {

const taskDate = new Date(task.created\_at);

const now = new Date();

const hoursDiff = (now - taskDate) / (1000 \* 60 \* 60);

return hoursDiff <= 24;

};

const priorityConfig = {

low: { bg: 'bg-green-100', text: 'text-green-700', label: 'Bassa' },

normal: { bg: 'bg-blue-100', text: 'text-blue-700', label: 'Normale' },

high: { bg: 'bg-orange-100', text: 'text-orange-700', label: 'Alta' },

urgent: { bg: 'bg-red-100', text: 'text-red-700', label: 'Urgente' }

};

const priority = priorityConfig[task.priority || 'normal'] || priorityConfig.normal;

const handleSaveEdit = async (field) => {

if (editValues[field] !== task[field]) {

await onInlineUpdate(task.id, { [field]: editValues[field] });

}

setEditingField(null);

};

const handleCancelEdit = () => {

setEditValues({

title: task.title,

description: task.description || '',

due\_date: task.due\_date

});

setEditingField(null);

};

const handleKeyDown = (e, field) => {

if (e.key === 'Enter') {

e.preventDefault();

handleSaveEdit(field);

} else if (e.key === 'Escape') {

handleCancelEdit();

}

};

const handleUpdate = async (field, value) => {

await onInlineUpdate(task.id, { [field]: value });

setEditingField(null);

};

const statusIcon = task.status === 'completed'

? <CheckCircle className="w-5 h-5 text-green-500 cursor-pointer" />

: <CheckCircle className="w-5 h-5 text-gray-300 cursor-pointer hover:text-gray-400" />;

// NUOVO: Classi per highlight righe

const rowClasses = `

border-b transition-all cursor-move

${isSelected ? 'bg-blue-50 hover:bg-blue-100' : 'hover:bg-gray-50'}

${isDragging ? 'opacity-50' : ''}

${editingField ? 'ring-2 ring-blue-400 ring-inset' : ''}

`;

return (

<tr className={rowClasses}>

{/\* NUOVO: Grip handle per drag & drop \*/}

<td className="p-2 w-8" {...dragHandleProps}>

<GripVertical className="w-4 h-4 text-gray-400 cursor-grab active:cursor-grabbing grip-handle" />

</td>

<td className="p-3">

<input

type="checkbox"

checked={isSelected}

onChange={onToggleSelect}

className="rounded text-blue-600"

/>

</td>

<td className="p-3">

<div onClick={() => onStatusChange(task.id, task.status === 'completed' ? 'pending' : 'completed')}>

{statusIcon}

</div>

</td>

{/\* Campo TITOLO - modificabile inline \*/}

<td className="p-3">

{editingField === 'title' ? (

<div className="flex items-center gap-2">

<input

ref={inputRef}

type="text"

value={editValues.title}

onChange={(e) => setEditValues({...editValues, title: e.target.value})}

onKeyDown={(e) => handleKeyDown(e, 'title')}

onBlur={() => handleSaveEdit('title')}

className="flex-1 px-2 py-1 border rounded focus:ring-2 focus:ring-blue-500"

/>

<button

onClick={() => handleSaveEdit('title')}

className="p-1 text-green-600 hover:bg-green-50 rounded"

>

<Check className="w-4 h-4" />

</button>

<button

onClick={handleCancelEdit}

className="p-1 text-red-600 hover:bg-red-50 rounded"

>

<X className="w-4 h-4" />

</button>

</div>

) : (

<div className="flex items-center gap-2 group">

<span

className="font-medium cursor-pointer hover:text-blue-600"

onClick={() => setEditingField('title')}

>

{highlightText(task.title, searchTerm)}

</span>

{isNewTask() && (

<span className="px-2 py-0.5 bg-green-100 text-green-700 text-xs rounded-full font-medium animate-pulse">

Nuovo

</span>

)}

<button

onClick={() => setEditingField('title')}

className="p-1 opacity-0 group-hover:opacity-100 transition-opacity text-gray-400 hover:text-gray-600"

>

<Edit2 className="w-3 h-3" />

</button>

</div>

)}

</td>

{/\* NUOVO: Colonna priorità con modifica inline dropdown \*/}

<td className="p-3">

{editingField === 'priority' ? (

<InlinePriorityEdit

priority={task.priority || 'normal'}

onSave={(value) => handleUpdate('priority', value)}

onCancel={() => setEditingField(null)}

/>

) : (

<div className="flex items-center gap-2 group">

<span

className={`px-2 py-1 ${priority.bg} ${priority.text} text-xs rounded-full font-medium cursor-pointer hover:ring-2 hover:ring-offset-1 hover:ring-blue-400`}

onClick={() => setEditingField('priority')}

>

{priority.label}

</span>

<button

onClick={() => setEditingField('priority')}

className="p-1 opacity-0 group-hover:opacity-100 transition-opacity text-gray-400 hover:text-gray-600"

>

<Edit2 className="w-3 h-3" />

</button>

</div>

)}

</td>

{/\* Campo DESCRIZIONE - modificabile inline \*/}

<td className="p-3 max-w-xs">

{editingField === 'description' ? (

<div className="flex items-center gap-2">

<input

ref={inputRef}

type="text"

value={editValues.description}

onChange={(e) => setEditValues({...editValues, description: e.target.value})}

onKeyDown={(e) => handleKeyDown(e, 'description')}

onBlur={() => handleSaveEdit('description')}

className="flex-1 px-2 py-1 border rounded focus:ring-2 focus:ring-blue-500"

/>

<button

onClick={() => handleSaveEdit('description')}

className="p-1 text-green-600 hover:bg-green-50 rounded"

>

<Check className="w-4 h-4" />

</button>

<button

onClick={handleCancelEdit}

className="p-1 text-red-600 hover:bg-red-50 rounded"

>

<X className="w-4 h-4" />

</button>

</div>

) : (

<div className="flex items-center gap-2 group">

<span

className="text-gray-600 truncate cursor-pointer hover:text-blue-600"

title={task.description}

onClick={() => setEditingField('description')}

>

{highlightText(task.description || '-', searchTerm)}

</span>

<button

onClick={() => setEditingField('description')}

className="p-1 opacity-0 group-hover:opacity-100 transition-opacity text-gray-400 hover:text-gray-600"

>

<Edit2 className="w-3 h-3" />

</button>

</div>

)}

</td>

{/\* Campo DATA - modificabile inline \*/}

<td className="p-3">

{editingField === 'due\_date' ? (

<div className="flex items-center gap-2">

<input

ref={inputRef}

type="date"

value={editValues.due\_date}

onChange={(e) => setEditValues({...editValues, due\_date: e.target.value})}

onKeyDown={(e) => handleKeyDown(e, 'due\_date')}

onBlur={() => handleSaveEdit('due\_date')}

className="px-2 py-1 border rounded focus:ring-2 focus:ring-blue-500"

/>

<button

onClick={() => handleSaveEdit('due\_date')}

className="p-1 text-green-600 hover:bg-green-50 rounded"

>

<Check className="w-4 h-4" />

</button>

<button

onClick={handleCancelEdit}

className="p-1 text-red-600 hover:bg-red-50 rounded"

>

<X className="w-4 h-4" />

</button>

</div>

) : (

<div className="flex items-center gap-2 group">

<span

className="text-sm cursor-pointer hover:text-blue-600"

onClick={() => setEditingField('due\_date')}

>

{formatDate(task.due\_date)}

</span>

<button

onClick={() => setEditingField('due\_date')}

className="p-1 opacity-0 group-hover:opacity-100 transition-opacity text-gray-400 hover:text-gray-600"

>

<Calendar className="w-3 h-3" />

</button>

</div>

)}

</td>

<td className="p-3">{highlightText(task.client || '-', searchTerm)}</td>

{/\* NUOVO: Colonna assegnatario con modifica inline dropdown \*/}

<td className="p-3">

{editingField === 'assignee' ? (

<InlineAssigneeEdit

assigneeId={task.assignee?.id}

assignees={assignees}

onSave={(value) => handleUpdate('assignee\_id', value)}

onCancel={() => setEditingField(null)}

/>

) : (

<div className="flex items-center gap-2 group">

{task.assignee ? (

<div

className="flex items-center gap-2 cursor-pointer hover:bg-gray-100 rounded-full pr-2"

onClick={() => setEditingField('assignee')}

>

<div className="w-8 h-8 bg-blue-500 text-white rounded-full flex items-center justify-center text-xs font-medium">

{task.assignee.name.substring(0, 2).toUpperCase()}

</div>

<span className="text-sm">{task.assignee.name}</span>

</div>

) : (

<button

onClick={() => setEditingField('assignee')}

className="flex items-center gap-2 px-3 py-1 text-sm text-gray-500 hover:text-gray-700 hover:bg-gray-100 rounded-full"

>

<User className="w-4 h-4" />

<span>Assegna</span>

</button>

)}

<button

onClick={() => setEditingField('assignee')}

className="p-1 opacity-0 group-hover:opacity-100 transition-opacity text-gray-400 hover:text-gray-600"

>

<Edit2 className="w-3 h-3" />

</button>

</div>

)}

</td>

<td className="p-3 max-w-xs truncate" title={task.opportunity}>

{highlightText(task.opportunity || '-', searchTerm)}

</td>

<td className="p-3">

<div className="flex -space-x-2">

<div className="w-8 h-8 bg-blue-500 text-white rounded-full flex items-center justify-center text-xs font-medium border-2 border-white">

GC

</div>

<div className="w-8 h-8 bg-blue-500 text-white rounded-full flex items-center justify-center text-xs font-medium border-2 border-white">

EF

</div>

</div>

</td>

<td className="p-3 text-sm">{formatDate(task.created\_at)}</td>

<td className="p-3 relative">

<button

onClick={() => setShowActions(!showActions)}

className="p-1 hover:bg-gray-100 rounded transition-colors"

>

<MoreVertical className="w-4 h-4" />

</button>

{showActions && (

<>

<div

className="fixed inset-0 z-10"

onClick={() => setShowActions(false)}

/>

<div className="absolute right-0 top-full mt-1 bg-white rounded-lg shadow-lg border py-1 z-20 min-w-[120px]">

<button

onClick={() => {

onEdit(task);

setShowActions(false);

}}

className="w-full px-4 py-2 text-left text-sm hover:bg-gray-100"

>

Modifica completa

</button>

<button

onClick={() => {

onDelete(task.id);

setShowActions(false);

}}

className="w-full px-4 py-2 text-left text-sm hover:bg-gray-100 text-red-600"

>

Elimina

</button>

</div>

</>

)}

</td>

</tr>

);

};

// 3.3 TaskTable - Componente principale tabella con tutte le funzionalità (MODIFICATO)

const TaskTable = ({

tasks = [],

loading = false,

onRefresh,

onTaskEdit,

onTaskDelete,

onTaskStatusChange,

onOpenFilters,

onInlineUpdate,

onTasksReorder,

assignees = []

}) => {

const [searchTerm, setSearchTerm] = useState('');

const [selectedTasks, setSelectedTasks] = useState([]);

const [currentView, setCurrentView] = useState('all');

const [sortConfig, setSortConfig] = useState({ key: null, direction: 'asc' });

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

const [taskFilter, setTaskFilter] = useState('all');

// NUOVO: Stati per drag & drop

const [draggedIndex, setDraggedIndex] = useState(null);

const [dragOverIndex, setDragOverIndex] = useState(null);

// 3.3.1 Calcolo contatori filtri rapidi

const quickFilterCounts = useMemo(() => {

const now = new Date();

const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());

const weekEnd = new Date(today);

weekEnd.setDate(weekEnd.getDate() + 7);

const monthEnd = new Date(today.getFullYear(), today.getMonth() + 1, 0);

return {

overdue: tasks.filter(t => new Date(t.due\_date) < today && t.status !== 'completed').length,

today: tasks.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate.toDateString() === today.toDateString();

}).length,

week: tasks.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate >= today && dueDate <= weekEnd;

}).length,

month: tasks.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate >= today && dueDate <= monthEnd;

}).length,

all: tasks.length

};

}, [tasks]);

// 3.3.2 Configurazione filtri rapidi

const quickFilters = [

{ id: 'overdue', label: 'Scadute', count: quickFilterCounts.overdue, color: 'bg-red-500' },

{ id: 'today', label: 'Oggi', count: quickFilterCounts.today, color: 'bg-yellow-500' },

{ id: 'week', label: 'Questa settimana', count: quickFilterCounts.week, color: 'bg-blue-500' },

{ id: 'month', label: 'Questo mese', count: quickFilterCounts.month, color: 'bg-green-500' },

{ id: 'all', label: 'Tutte', count: quickFilterCounts.all, color: 'bg-gray-500' }

];

// 3.3.3 Filtraggio e ordinamento task

const filteredTasks = useMemo(() => {

let filtered = [...tasks];

// Applica ricerca

if (searchTerm) {

filtered = filtered.filter(task =>

task.title.toLowerCase().includes(searchTerm.toLowerCase()) ||

task.description?.toLowerCase().includes(searchTerm.toLowerCase()) ||

task.client?.toLowerCase().includes(searchTerm.toLowerCase()) ||

task.opportunity?.toLowerCase().includes(searchTerm.toLowerCase())

);

}

// Applica filtri vista temporale

const now = new Date();

const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());

switch (currentView) {

case 'overdue':

filtered = filtered.filter(t => new Date(t.due\_date) < today && t.status !== 'completed');

break;

case 'today':

filtered = filtered.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate.toDateString() === today.toDateString();

});

break;

case 'week':

const weekEnd = new Date(today);

weekEnd.setDate(weekEnd.getDate() + 7);

filtered = filtered.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate >= today && dueDate <= weekEnd;

});

break;

case 'month':

const monthEnd = new Date(today.getFullYear(), today.getMonth() + 1, 0);

filtered = filtered.filter(t => {

const dueDate = new Date(t.due\_date);

return dueDate >= today && dueDate <= monthEnd;

});

break;

}

// Applica filtri assegnazione

switch (taskFilter) {

case 'mine':

filtered = filtered.filter(t => t.assignee?.id === 'current\_user');

break;

case 'assigned':

filtered = filtered.filter(t => t.assignee?.id !== 'current\_user');

break;

}

// Applica ordinamento

if (sortConfig.key) {

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

let aVal = a[sortConfig.key];

let bVal = b[sortConfig.key];

if (sortConfig.key === 'assignee') {

aVal = a.assignee?.name || '';

bVal = b.assignee?.name || '';

}

if (aVal < bVal) return sortConfig.direction === 'asc' ? -1 : 1;

if (aVal > bVal) return sortConfig.direction === 'asc' ? 1 : -1;

return 0;

});

}

return filtered;

}, [tasks, searchTerm, currentView, sortConfig, taskFilter]);

// 3.3.4 Handlers

const handleSort = (key) => {

setSortConfig(prev => ({

key,

direction: prev.key === key && prev.direction === 'asc' ? 'desc' : 'asc'

}));

};

const handleSelectAll = (checked) => {

if (checked) {

setSelectedTasks(filteredTasks.map(t => t.id));

} else {

setSelectedTasks([]);

}

};

const handleBulkAction = (action) => {

if (selectedTasks.length === 0) return;

switch (action) {

case 'delete':

if (window.confirm(`Eliminare ${selectedTasks.length} attività?`)) {

selectedTasks.forEach(id => onTaskDelete(id));

setSelectedTasks([]);

}

break;

case 'complete':

selectedTasks.forEach(id => onTaskStatusChange(id, 'completed'));

break;

case 'assign':

console.log('Assegnazione multipla');

break;

}

};

// NUOVO: Handlers drag & drop

const handleDragStart = (e, index) => {

setDraggedIndex(index);

e.dataTransfer.effectAllowed = 'move';

};

const handleDragOver = (e, index) => {

e.preventDefault();

if (index !== dragOverIndex) {

setDragOverIndex(index);

}

};

const handleDrop = async (e, dropIndex) => {

e.preventDefault();

if (draggedIndex === null || draggedIndex === dropIndex) {

setDraggedIndex(null);

setDragOverIndex(null);

return;

}

const newTasks = [...filteredTasks];

const draggedTask = newTasks[draggedIndex];

newTasks.splice(draggedIndex, 1);

newTasks.splice(dropIndex, 0, draggedTask);

await onTasksReorder(newTasks);

setDraggedIndex(null);

setDragOverIndex(null);

};

const handleDragEnd = () => {

setDraggedIndex(null);

setDragOverIndex(null);

};

// 3.3.5 Render componente

return (

<div>

{/\* Toolbar con filtri e ricerca \*/}

<div className="px-6 py-4 bg-white border-b">

{/\* Intestazione Elenco Attività con statistiche \*/}

<div className="flex items-center justify-between mb-4">

<div>

<h2 className="text-xl font-semibold text-gray-800">Elenco Attività</h2>

<p className="text-sm text-gray-600 mt-1">

{tasks.length} attività totali • {tasks.filter(t => t.status === 'completed').length} completate • {tasks.filter(t => t.status === 'pending').length} in sospeso

</p>

</div>

<div className="flex items-center gap-2">

<button className="px-4 py-2 text-sm border rounded-lg hover:bg-gray-50 flex items-center gap-2">

<Download className="w-4 h-4" />

Esporta

</button>

<button className="px-4 py-2 text-sm border rounded-lg hover:bg-gray-50 flex items-center gap-2">

<FileText className="w-4 h-4" />

Report

</button>

</div>

</div>

{/\* Filtri e ricerca \*/}

<div className="flex items-center justify-between mb-4">

{/\* Filtri rapidi \*/}

<div className="flex gap-2 overflow-x-auto pb-2">

{quickFilters.map(filter => (

<button

key={filter.id}

onClick={() => setCurrentView(filter.id)}

className={`px-4 py-2 rounded-lg text-sm font-medium transition-all whitespace-nowrap ${

currentView === filter.id

? `${filter.color} text-white`

: 'bg-gray-100 text-gray-700 hover:bg-gray-200'

}`}

>

{filter.label} ({filter.count})

</button>

))}

</div>

{/\* Barra ricerca e azioni \*/}

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

<div className="relative">

<Search className="w-5 h-5 absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400" />

<input

type="text"

placeholder="Filtra le informazioni"

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

className="pl-10 pr-10 py-2 border rounded-lg w-80 focus:ring-2 focus:ring-blue-500"

/>

<button

onClick={onOpenFilters}

className="absolute right-2 top-1/2 -translate-y-1/2 p-1 hover:bg-gray-100 rounded"

title="Filtri avanzati"

>

<ChevronDown className="w-4 h-4" />

</button>

</div>

<button

onClick={onRefresh}

disabled={loading}

className="p-2 hover:bg-gray-100 rounded-lg transition-colors"

title="Aggiorna"

>

<RefreshCw className={`w-5 h-5 ${loading ? 'animate-spin' : ''}`} />

</button>

<div className="flex gap-2">

<select

value={taskFilter}

onChange={(e) => setTaskFilter(e.target.value)}

className="px-4 py-2 border rounded-lg bg-white"

>

<option value="all">Tutte le attività</option>

<option value="mine">Le mie attività</option>

<option value="assigned">Attività assegnate</option>

</select>

<select

value={viewMode}

onChange={(e) => setViewMode(e.target.value)}

className="px-4 py-2 border rounded-lg bg-white"

>

<option value="list">Elenco</option>

<option value="kanban">Kanban</option>

</select>

</div>

</div>

</div>

{/\* Azioni bulk \*/}

{selectedTasks.length > 0 && (

<div className="flex items-center gap-4 p-3 bg-blue-50 rounded-lg animate-fadeIn">

<span className="text-sm font-medium">

{selectedTasks.length} attività selezionate

</span>

<button

onClick={() => handleBulkAction('assign')}

className="text-sm text-blue-600 hover:text-blue-800"

>

Assegna

</button>

<button

onClick={() => handleBulkAction('complete')}

className="text-sm text-blue-600 hover:text-blue-800"

>

Completa

</button>

<button

onClick={() => handleBulkAction('delete')}

className="text-sm text-red-600 hover:text-red-800"

>

Elimina

</button>

<button

onClick={() => setSelectedTasks([])}

className="ml-auto text-sm text-gray-600 hover:text-gray-800"

>

<X className="w-4 h-4" />

</button>

</div>

)}

{/\* NUOVO: Indicatore ricerca attiva \*/}

{searchTerm && (

<div className="mt-2 text-sm text-gray-600 flex items-center gap-2">

<Search className="w-4 h-4" />

<span>

Trovate <strong>{filteredTasks.length}</strong> attività per "{searchTerm}"

</span>

<button

onClick={() => setSearchTerm('')}

className="text-blue-600 hover:text-blue-800 underline"

>

Cancella ricerca

</button>

</div>

)}

</div>

{/\* Tabella \*/}

<div className="px-6 py-4">

{/\* Intestazione della sezione tabella con statistiche \*/}

<div className="mb-6 flex items-center justify-between">

<div>

<h2 className="text-xl font-semibold text-gray-800">Elenco Attività</h2>

<p className="text-sm text-gray-600 mt-1">

{tasks.length} attività totali • {tasks.filter(t => t.status === 'completed').length} completate • {tasks.filter(t => t.status === 'pending').length} in sospeso

</p>

</div>

<div className="flex items-center gap-2">

<button className="px-4 py-2 text-sm border rounded-lg hover:bg-gray-50 flex items-center gap-2">

<Download className="w-4 h-4" />

Esporta

</button>

<button className="px-4 py-2 text-sm border rounded-lg hover:bg-gray-50 flex items-center gap-2">

<FileText className="w-4 h-4" />

Report

</button>

</div>

</div>

<div className="bg-white rounded-lg shadow-sm border overflow-hidden">

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

<table className="w-full">

{/\* Barra azioni sopra le intestazioni \*/}

<thead>

<tr className="bg-gray-100 border-b">

<th colSpan={13} className="px-3 py-2">

<div className="flex items-center justify-between">

<div className="flex items-center gap-3">

<span className="text-sm font-medium text-gray-600">AZIONI:</span>

<select

className="text-sm border rounded px-2 py-1 bg-white"

onChange={(e) => {

if (e.target.value === 'all') {

handleSelectAll(true);

} else if (e.target.value === 'none') {

handleSelectAll(false);

}

e.target.value = '';

}}

>

<option value="">Seleziona...</option>

<option value="all">Seleziona tutti</option>

<option value="none">Deseleziona tutti</option>

</select>

</div>

<div className="flex items-center gap-2 text-sm">

<span className="text-gray-500">

{selectedTasks.length > 0 && `${selectedTasks.length} selezionati`}

</span>

</div>

</div>

</th>

</tr>

{/\* Intestazioni colonne \*/}

<tr className="bg-gray-50 border-b">

<th className="p-2 w-8">

{/\* Colonna drag handle \*/}

</th>

<th className="p-3 text-left">

<input

type="checkbox"

checked={selectedTasks.length === filteredTasks.length && filteredTasks.length > 0}

onChange={(e) => handleSelectAll(e.target.checked)}

className="rounded text-blue-600"

title="Seleziona tutto"

/>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('status')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

title="Stato"

>

Stato

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('title')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Titolo

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('priority')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Priorità

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('description')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Descrizione

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('due\_date')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Scadenza

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('client')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Cliente

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('assignee')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Assegnato a

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('opportunity')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Opportunità

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-left">

<span className="font-medium text-xs uppercase">

Commenti

</span>

</th>

<th className="p-3 text-left">

<button

onClick={() => handleSort('created\_at')}

className="flex items-center gap-1 hover:text-gray-900 font-medium text-xs uppercase"

>

Creato

<ArrowUpDown className="w-3 h-3" />

</button>

</th>

<th className="p-3 text-center">

<button

className="p-1 hover:bg-gray-100 rounded"

title="Gestisci colonne"

>

<MoreVertical className="w-4 h-4" />

</button>

</th>

</tr>

</thead>

<tbody>

{loading ? (

// Loading state

Array(5).fill(0).map((\_, i) => <TaskSkeleton key={i} />)

) : filteredTasks.length === 0 ? (

// Empty state

<tr>

<td colSpan={13} className="p-8 text-center">

<FileText className="w-12 h-12 text-gray-300 mx-auto mb-2" />

<p className="text-gray-500">Nessuna attività trovata</p>

<p className="text-sm text-gray-400 mt-1">

Prova a modificare i filtri o aggiungi una nuova attività

</p>

</td>

</tr>

) : (

// Task rows con drag & drop

filteredTasks.map((task, index) => (

<tr

key={task.id}

draggable

onDragStart={(e) => handleDragStart(e, index)}

onDragOver={(e) => handleDragOver(e, index)}

onDrop={(e) => handleDrop(e, index)}

onDragEnd={handleDragEnd}

className={dragOverIndex === index ? 'border-t-2 border-blue-500' : ''}

>

<TaskRow

task={task}

isSelected={selectedTasks.includes(task.id)}

onToggleSelect={() => {

if (selectedTasks.includes(task.id)) {

setSelectedTasks(selectedTasks.filter(id => id !== task.id));

} else {

setSelectedTasks([...selectedTasks, task.id]);

}

}}

onEdit={onTaskEdit}

onDelete={onTaskDelete}

onStatusChange={onTaskStatusChange}

onInlineUpdate={onInlineUpdate}

searchTerm={searchTerm}

assignees={assignees}

isDragging={draggedIndex === index}

dragHandleProps={{}}

/>

</tr>

))

)}

</tbody>

</table>

</div>

</div>

{/\* Info paginazione \*/}

{filteredTasks.length > 0 && (

<div className="mt-4 flex items-center justify-between text-sm text-gray-600">

<p>

Visualizzando {filteredTasks.length} di {tasks.length} attività

</p>

</div>

)}

</div>

{/\* Indicatore visivo durante il drag \*/}

{draggedIndex !== null && (

<div className="fixed bottom-4 right-4 bg-blue-600 text-white px-4 py-2 rounded-lg shadow-lg">

Trascina per riordinare

</div>

)}

</div>

);

};

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

SEZIONE 4: ADVANCED FILTERS (Componente C)

- Pannello filtri avanzati

- 3 Tab: Filtri, Personalizzati, Preferiti

- Gestione filtri salvati

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

// 4.1 FilterSection - Sezione filtro espandibile

const FilterSection = ({

title,

isExpanded,

onToggle,

children

}) => {

return (

<div className="border rounded-lg">

<button

onClick={onToggle}

className="w-full px-4 py-3 flex items-center justify-between hover:bg-gray-50 transition-colors"

>

<span className="font-medium">{title}</span>

<ChevronRight className={`w-4 h-4 transition-transform ${isExpanded ? 'rotate-90' : ''}`} />

</button>

{isExpanded && (

<div className="px-4 pb-3 border-t animate-slideDown">

<div className="mt-3">

{children}

</div>

</div>

)}

</div>

);

};

// 4.2 AdvancedFilters - Componente principale filtri

const AdvancedFilters = ({

isOpen,

onClose,

filters = {},

onFiltersChange,

onSaveFilter,

assignees = [],

clients = [],

groups = [],

projects = []

}) => {

const [activeTab, setActiveTab] = useState('filters');

const [localFilters, setLocalFilters] = useState(filters);

const [filterName, setFilterName] = useState('');

const [showSaveDialog, setShowSaveDialog] = useState(false);

const [savedFilters, setSavedFilters] = useState([]);

const [favoriteFilters, setFavoriteFilters] = useState([]);

const [searchTerm, setSearchTerm] = useState('');

// 4.2.1 Stati sezioni espanse

const [expandedSections, setExpandedSections] = useState({

statoAttivita: true,

assegnatario: true,

utenteSingolo: false,

utenteGruppo: false,

dataScadenza: false,

dataInizio: false,

collegatoA: false

});

// 4.2.2 Mock filtri salvati

useEffect(() => {

setSavedFilters([

{ id: 1, name: 'Ass. Niccolò', filters: { assignee: 'niccolo' } },

{ id: 2, name: 'Ass. \_ Mario', filters: { assignee: 'mario' } },

{ id: 3, name: 'Ass. \_ Noemi', filters: { assignee: 'noemi' } },

{ id: 4, name: 'Ass. a \_ Niccolò', filters: { assignee: 'niccolo', status: 'pending' } },

{ id: 5, name: 'Ass. a \_ Flagiello', filters: { assignee: 'flagiello' } },

{ id: 6, name: 'Ass. Da Me \_ Non Completato', filters: { assignee: 'me', status: 'incomplete' } }

]);

}, []);

// 4.2.3 Handlers

const toggleSection = (section) => {

setExpandedSections(prev => ({

...prev,

[section]: !prev[section]

}));

};

const handleApply = () => {

onFiltersChange(localFilters);

onClose();

};

const handleReset = () => {

setLocalFilters({});

};

const handleSaveFilter = () => {

if (filterName.trim()) {

const newFilter = {

id: Date.now(),

name: filterName,

filters: localFilters,

created\_at: new Date().toISOString()

};

setSavedFilters([...savedFilters, newFilter]);

onSaveFilter(newFilter);

setShowSaveDialog(false);

setFilterName('');

}

};

const applySavedFilter = (savedFilter) => {

setLocalFilters(savedFilter.filters);

setActiveTab('filters');

};

const toggleFavorite = (filterId) => {

if (favoriteFilters.includes(filterId)) {

setFavoriteFilters(favoriteFilters.filter(id => id !== filterId));

} else {

setFavoriteFilters([...favoriteFilters, filterId]);

}

};

if (!isOpen) return null;

// 4.2.4 Render componente

return (

<div className="fixed inset-0 bg-black bg-opacity-50 z-50 flex items-center justify-center">

<div className="bg-white rounded-lg shadow-xl w-full max-w-5xl max-h-[90vh] overflow-hidden">

{/\* Header con ricerca \*/}

<div className="px-6 py-4 border-b bg-gray-50">

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

<div className="relative flex-1">

<Search className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400" />

<input

type="text"

placeholder="Filtra le informazioni"

value={searchTerm}

onChange={(e) => setSearchTerm(e.target.value)}

className="w-full pl-10 pr-10 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

/>

<ChevronDown className="absolute right-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400" />

</div>

<button onClick={onClose} className="text-gray-400 hover:text-gray-600">

<X className="w-6 h-6" />

</button>

</div>

</div>

{/\* Tabs \*/}

<div className="flex border-b bg-white">

<button

onClick={() => setActiveTab('filters')}

className={`px-6 py-3 font-medium flex items-center gap-2 border-b-2 transition-colors ${

activeTab === 'filters'

? 'text-blue-600 border-blue-600'

: 'text-gray-600 border-transparent hover:text-gray-800'

}`}

>

<Filter className="w-4 h-4" />

Filtri

</button>

<button

onClick={() => setActiveTab('custom')}

className={`px-6 py-3 font-medium flex items-center gap-2 border-b-2 transition-colors ${

activeTab === 'custom'

? 'text-green-600 border-green-600'

: 'text-gray-600 border-transparent hover:text-gray-800'

}`}

>

<FolderOpen className="w-4 h-4" />

Filtri personalizzati

</button>

<button

onClick={() => setActiveTab('favorites')}

className={`px-6 py-3 font-medium flex items-center gap-2 border-b-2 transition-colors ${

activeTab === 'favorites'

? 'text-yellow-600 border-yellow-600'

: 'text-gray-600 border-transparent hover:text-gray-800'

}`}

>

<Star className="w-4 h-4" />

Preferiti

</button>

</div>

{/\* Content \*/}

<div className="p-6 overflow-y-auto" style={{ maxHeight: 'calc(90vh - 200px)' }}>

{/\* TAB FILTRI \*/}

{activeTab === 'filters' && (

<div className="grid grid-cols-2 gap-6">

{/\* Colonna sinistra \*/}

<div className="space-y-4">

{/\* Stato Attività \*/}

<FilterSection

title="Stato Attività"

isExpanded={expandedSections.statoAttivita}

onToggle={() => toggleSection('statoAttivita')}

>

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

<label className="flex items-center gap-2 cursor-pointer">

<input

type="checkbox"

className="rounded text-blue-600"

checked={localFilters.status?.includes('open')}

onChange={(e) => {

const statuses = localFilters.status || [];

if (e.target.checked) {

setLocalFilters({...localFilters, status: [...statuses, 'open']});

} else {

setLocalFilters({...localFilters, status: statuses.filter(s => s !== 'open')});

}

}}

/>

<span className="text-sm">Aperto</span>

</label>

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">In Corso</span>

</label>

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">Completato</span>

</label>

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">Annullato</span>

</label>

</div>

</FilterSection>

{/\* Assegnatario \*/}

<FilterSection

title="Assegnatario"

isExpanded={expandedSections.assegnatario}

onToggle={() => toggleSection('assegnatario')}

>

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

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">Assegnato a me</span>

</label>

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">Non assegnato</span>

</label>

<label className="flex items-center gap-2 cursor-pointer">

<input type="checkbox" className="rounded text-blue-600" />

<span className="text-sm">Assegnato ad altri</span>

</label>

</div>

</FilterSection>

{/\* Utente Singolo \*/}

<FilterSection

title="Utente Singolo"

isExpanded={expandedSections.utenteSingolo}

onToggle={() => toggleSection('utenteSingolo')}

>

<SearchableSelect

placeholder="Seleziona utente..."

value={localFilters.utenteSingolo}

onChange={(value) => setLocalFilters({...localFilters, utenteSingolo: value})}

options={assignees}

/>

</FilterSection>

{/\* Utente Gruppo \*/}

<FilterSection

title="Utente Gruppo"

isExpanded={expandedSections.utenteGruppo}

onToggle={() => toggleSection('utenteGruppo')}

>

<SearchableSelect

placeholder="Seleziona gruppo..."

value={localFilters.utenteGruppo}

onChange={(value) => setLocalFilters({...localFilters, utenteGruppo: value})}

options={groups}

/>

</FilterSection>

</div>

{/\* Colonna destra \*/}

<div className="space-y-4">

{/\* Data Scadenza \*/}

<FilterSection

title="Data Scadenza"

isExpanded={expandedSections.dataScadenza}

onToggle={() => toggleSection('dataScadenza')}

>

<div className="grid grid-cols-2 gap-2">

<input

type="date"

className="px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

placeholder="Da"

value={localFilters.dueDateFrom || ''}

onChange={(e) => setLocalFilters({...localFilters, dueDateFrom: e.target.value})}

/>

<input

type="date"

className="px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

placeholder="A"

value={localFilters.dueDateTo || ''}

onChange={(e) => setLocalFilters({...localFilters, dueDateTo: e.target.value})}

/>

</div>

</FilterSection>

{/\* Data Inizio \*/}

<FilterSection

title="Data Inizio"

isExpanded={expandedSections.dataInizio}

onToggle={() => toggleSection('dataInizio')}

>

<div className="grid grid-cols-2 gap-2">

<input

type="date"

className="px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

placeholder="Da"

value={localFilters.startDateFrom || ''}

onChange={(e) => setLocalFilters({...localFilters, startDateFrom: e.target.value})}

/>

<input

type="date"

className="px-3 py-2 border rounded-lg focus:ring-2 focus:ring-blue-500"

placeholder="A"

value={localFilters.startDateTo || ''}

onChange={(e) => setLocalFilters({...localFilters, startDateTo: e.target.value})}

/>

</div>

</FilterSection>

{/\* Collegato A \*/}

<FilterSection

title="Collegato A"

isExpanded={expandedSections.collegatoA}

onToggle={() => toggleSection('collegatoA')}

>

<SearchableSelect

placeholder="Seleziona collegamento..."

value={localFilters.collegatoA}

onChange={(value) => setLocalFilters({...localFilters, collegatoA: value})}

options={projects}

/>

</FilterSection>

{/\* Salva ricerca corrente \*/}

<button

onClick={() => setShowSaveDialog(true)}

className="w-full px-4 py-3 text-left hover:bg-gray-100 rounded-lg flex items-center gap-2 transition-colors"

>

<ChevronRight className="w-4 h-4" />

<span className="font-medium">Salva ricerca corrente</span>

</button>

</div>

</div>

)}

{/\* TAB FILTRI PERSONALIZZATI \*/}

{activeTab === 'custom' && (

<div className="space-y-4">

{savedFilters.length > 0 ? (

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

{savedFilters.map(filter => (

<div

key={filter.id}

className="border rounded-lg p-4 hover:bg-gray-50 transition-colors"

>

<div className="flex items-center justify-between mb-2">

<h4 className="font-medium">{filter.name}</h4>

<div className="flex gap-2">

<button

onClick={() => toggleFavorite(filter.id)}

className={`p-1 rounded hover:bg-gray-200 ${

favoriteFilters.includes(filter.id) ? 'text-yellow-500' : 'text-gray-400'

}`}

>

<Star className="w-4 h-4" fill={favoriteFilters.includes(filter.id) ? 'currentColor' : 'none'} />

</button>

<button

onClick={() => {

if (window.confirm('Eliminare questo filtro?')) {

setSavedFilters(savedFilters.filter(f => f.id !== filter.id));

}

}}

className="p-1 rounded hover:bg-gray-200 text-red-500"

>

<X className="w-4 h-4" />

</button>

</div>

</div>

<p className="text-sm text-gray-600 mb-3">

{Object.keys(filter.filters).length} criteri applicati

</p>

<button

onClick={() => applySavedFilter(filter)}

className="w-full px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-700 text-sm"

>

Applica filtro

</button>

</div>

))}

</div>

) : (

<div className="text-center py-8">

<FolderOpen className="w-12 h-12 text-gray-300 mx-auto mb-3" />

<p className="text-gray-500">Nessun filtro personalizzato salvato</p>

<button

onClick={() => setActiveTab('filters')}

className="mt-3 text-blue-600 hover:text-blue-800 text-sm"

>

Crea il tuo primo filtro

</button>

</div>

)}

</div>

)}

{/\* TAB PREFERITI \*/}

{activeTab === 'favorites' && (

<div className="space-y-4">

{favoriteFilters.length > 0 ? (

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

{savedFilters

.filter(f => favoriteFilters.includes(f.id))

.map(filter => (

<div

key={filter.id}

className="border rounded-lg p-4 hover:bg-gray-50 transition-colors"

>

<div className="flex items-center justify-between mb-2">

<h4 className="font-medium">{filter.name}</h4>

<button

onClick={() => toggleFavorite(filter.id)}

className="p-1 rounded hover:bg-gray-200 text-yellow-500"

>

<Star className="w-4 h-4" fill="currentColor" />

</button>

</div>

<button

onClick={() => applySavedFilter(filter)}

className="w-full px-3 py-2 bg-blue-600 text-white rounded hover:bg-blue-700 text-sm"

>

Applica filtro

</button>

</div>

))}

</div>

) : (

<div className="text-center py-8">

<Star className="w-12 h-12 text-gray-300 mx-auto mb-3" />

<p className="text-gray-500">Nessun filtro preferito</p>

<p className="text-sm text-gray-400 mt-1">

Aggiungi filtri ai preferiti per accedervi rapidamente

</p>

</div>

)}

</div>

)}

</div>

{/\* Footer con azioni \*/}

<div className="px-6 py-4 border-t bg-gray-50 flex justify-between">

<button

onClick={handleReset}

className="px-4 py-2 text-gray-600 hover:text-gray-800 transition-colors"

>

Resetta filtri

</button>

<div className="space-x-2">

<button

onClick={onClose}

className="px-4 py-2 text-gray-600 hover:text-gray-800 transition-colors"

>

Annulla

</button>

<button

onClick={handleApply}

className="px-6 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors"

>

Applica filtri

</button>

</div>

</div>

{/\* Dialog salva filtro \*/}

{showSaveDialog && (

<div className="absolute inset-0 bg-black bg-opacity-50 flex items-center justify-center">

<div className="bg-white rounded-lg p-6 w-96">

<h3 className="text-lg font-semibold mb-4">Salva filtro personalizzato</h3>

<input

type="text"

placeholder="Nome del filtro..."

value={filterName}

onChange={(e) => setFilterName(e.target.value)}

className="w-full px-3 py-2 border rounded-lg mb-4 focus:ring-2 focus:ring-blue-500"

autoFocus

/>

<div className="flex justify-end gap-2">

<button

onClick={() => {

setShowSaveDialog(false);

setFilterName('');

}}

className="px-4 py-2 text-gray-600 hover:text-gray-800"

>

Annulla

</button>

<button

onClick={handleSaveFilter}

disabled={!filterName.trim()}

className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 disabled:opacity-50 disabled:cursor-not-allowed"

>

Salva

</button>

</div>

</div>

</div>

)}

</div>

</div>

);

};

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

SEZIONE 5: MAIN COMPONENT

- Orchestratore principale

- Gestione stato globale

- Mock data

- Handlers principali

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

export default function TaskManagement() {

// 5.1 Stati principali

const [tasks, setTasks] = useState([]);

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

const [showFilters, setShowFilters] = useState(false);

const [toast, setToast] = useState(null);

const [taskGroup, setTaskGroup] = useState('tasks');

// 5.2 Mock data per dropdown

const assignees = [

{ value: 1, label: 'EF' },

{ value: 2, label: 'GC' },

{ value: 3, label: 'Mario' },

{ value: 4, label: 'Niccolò' },

{ value: 5, label: 'Ass. Niccolò' },

{ value: 6, label: 'Ass. \_ Mario' },

{ value: 7, label: 'Ass. \_ Noemi' },

{ value: 8, label: 'Ass. a \_ Niccolò' },

{ value: 9, label: 'Ass. a \_ Flagiello' },

{ value: 10, label: 'Ass. Da Me \_ Non Completato' }

];

const clients = [

{ value: 1, label: 'Minone pino (casa sua)' },

{ value: 2, label: 'Azienda ABC' },

{ value: 3, label: 'Studio Legale XYZ' },

{ value: 4, label: 'Consulting Group' },

{ value: 5, label: 'Tech Solutions' }

];

const contacts = [

{ value: 1, label: 'Mario Rossi' },

{ value: 2, label: 'Giuseppe Verdi' },

{ value: 3, label: 'Anna Bianchi' },

{ value: 4, label: 'Luca Neri' },

{ value: 5, label: 'Sofia Romano' }

];

const groups = [

{ value: 1, label: 'Amministrazione' },

{ value: 2, label: 'Commerciale' },

{ value: 3, label: 'Sviluppo' },

{ value: 4, label: 'Marketing' }

];

const projects = [

{ value: 1, label: 'Progetto Alpha' },

{ value: 2, label: 'Progetto Beta' },

{ value: 3, label: 'Progetto Gamma' }

];

// 5.3 Caricamento iniziale

useEffect(() => {

loadTasks();

}, []);

// 5.4 Funzione caricamento task

const loadTasks = async () => {

setLoading(true);

try {

await new Promise(resolve => setTimeout(resolve, 500));

const mockTasks = [

{

id: 1,

title: 'App x Sic ecc.',

description: 'Viene x parlare del da farsi',

status: 'completed',

priority: 'high',

due\_date: '2022-10-28',

client: 'Minone pino (casa sua)',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 100,

created\_at: '2022-10-28'

},

{

id: 2,

title: 'Chiamare all\'amministratore',

description: '',

status: 'pending',

priority: 'urgent',

due\_date: '2022-11-23',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: '2022-11-23'

},

{

id: 3,

title: 'Contatto con il cliente',

description: '',

status: 'pending',

priority: 'normal',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: new Date().toISOString()

},

{

id: 4,

title: 'Attesa documentazione da parte del cliente',

description: '',

status: 'pending',

priority: 'low',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: '2023-02-28'

},

{

id: 5,

title: 'Controllo documentazione ricevuta',

description: '',

status: 'pending',

priority: 'normal',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: '2023-02-28'

},

{

id: 6,

title: 'Lavorazione pratica',

description: '',

status: 'pending',

priority: 'high',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: '2023-02-28'

},

{

id: 7,

title: 'Review',

description: '',

status: 'pending',

priority: 'normal',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: new Date(Date.now() - 2 \* 60 \* 60 \* 1000).toISOString()

},

{

id: 8,

title: 'Pianificazione formazione',

description: '',

status: 'pending',

priority: 'low',

due\_date: '2023-02-28',

client: '',

assignee: { id: 1, name: 'EF' },

opportunity: '\*\* QUALIFICA GR...',

completion: 0,

created\_at: '2023-02-28'

}

];

setTasks(mockTasks);

} catch (error) {

console.error('Errore caricamento tasks:', error);

setToast({ message: 'Errore nel caricamento delle attività', type: 'error' });

} finally {

setLoading(false);

}

};

// 5.5 Handler aggiunta task

const handleAddTask = async (taskData) => {

try {

await new Promise(resolve => setTimeout(resolve, 500));

const newTask = {

...taskData,

id: Date.now(),

status: taskData.status || 'pending',

priority: taskData.priority || 'normal',

completion: 0,

created\_at: taskData.created\_at || new Date().toISOString(),

assignee: taskData.assignees?.[0] ?

assignees.find(a => a.value === taskData.assignees[0]) || null :

null

};

setTasks([newTask, ...tasks]);

setToast({ message: 'Attività aggiunta con successo', type: 'success' });

return true;

} catch (error) {

setToast({ message: 'Errore nell\'aggiunta dell\'attività', type: 'error' });

throw error;

}

};

// 5.6 Handler modifica inline (NUOVO)

const handleInlineUpdate = async (taskId, updates) => {

try {

await new Promise(resolve => setTimeout(resolve, 300));

setTasks(tasks.map(task => {

if (task.id === taskId) {

// Se aggiorniamo l'assignee\_id, dobbiamo anche aggiornare l'oggetto assignee

if (updates.assignee\_id) {

const newAssignee = assignees.find(a => a.value === updates.assignee\_id);

return {

...task,

...updates,

assignee: newAssignee ? { id: newAssignee.value, name: newAssignee.label } : null

};

}

return { ...task, ...updates };

}

return task;

}));

setToast({ message: 'Attività aggiornata', type: 'success' });

} catch (error) {

setToast({ message: 'Errore nell\'aggiornamento', type: 'error' });

}

};

// 5.7 Handler riordinamento task (NUOVO)

const handleTasksReorder = async (reorderedTasks) => {

try {

await new Promise(resolve => setTimeout(resolve, 500));

setTasks(reorderedTasks);

setToast({ message: 'Ordine aggiornato', type: 'success' });

return true;

} catch (error) {

setToast({ message: 'Errore nel riordinamento', type: 'error' });

throw error;

}

};

// 5.8 Altri handlers

const handleTaskEdit = (task) => {

console.log('Edit task:', task);

setToast({ message: 'Funzionalità modifica completa in sviluppo', type: 'info' });

};

const handleTaskDelete = async (taskId) => {

if (window.confirm('Sei sicuro di voler eliminare questa attività?')) {

try {

await new Promise(resolve => setTimeout(resolve, 300));

setTasks(tasks.filter(t => t.id !== taskId));

setToast({ message: 'Attività eliminata', type: 'success' });

} catch (error) {

setToast({ message: 'Errore nell\'eliminazione', type: 'error' });

}

}

};

const handleTaskStatusChange = async (taskId, newStatus) => {

try {

await new Promise(resolve => setTimeout(resolve, 300));

setTasks(tasks.map(task =>

task.id === taskId

? { ...task, status: newStatus, completion: newStatus === 'completed' ? 100 : task.completion }

: task

));

setToast({

message: newStatus === 'completed' ? 'Attività completata' : 'Stato aggiornato',

type: 'success'

});

} catch (error) {

setToast({ message: 'Errore nell\'aggiornamento', type: 'error' });

}

};

const handleFiltersChange = (filters) => {

console.log('Filtri applicati:', filters);

setShowFilters(false);

setToast({ message: 'Filtri applicati', type: 'success' });

};

const handleSaveFilter = (filter) => {

console.log('Filtro salvato:', filter);

localStorage.setItem(`filter\_${filter.id}`, JSON.stringify(filter));

};

// 5.9 Render principale

return (

<div className="min-h-screen bg-gray-50">

{/\* Header \*/}

<div className="bg-white border-b px-6 py-4 sticky top-0 z-40">

<div className="flex items-center justify-between">

<h1 className="text-2xl font-semibold text-gray-800">Elenco Attività</h1>

<div className="flex items-center gap-2">

<button

onClick={() => setTaskGroup('tasks')}

className={`px-4 py-2 font-medium transition-colors ${

taskGroup === 'tasks'

? 'text-blue-600 border-b-2 border-blue-600'

: 'text-gray-600 hover:text-gray-800'

}`}

>

ATTIVITÀ

</button>

<button

onClick={() => setTaskGroup('projects')}

className={`px-4 py-2 font-medium transition-colors ${

taskGroup === 'projects'

? 'text-blue-600 border-b-2 border-blue-600'

: 'text-gray-600 hover:text-gray-800'

}`}

>

PROGETTI

</button>

</div>

</div>

</div>

{/\* Quick Add Task \*/}

<div className="px-6 py-4 bg-white border-b">

<QuickAddTask

onAdd={handleAddTask}

assignees={assignees}

clients={clients}

contacts={contacts}

/>

</div>

{/\* Task Table \*/}

<TaskTable

tasks={tasks}

loading={loading}

onRefresh={loadTasks}

onTaskEdit={handleTaskEdit}

onTaskDelete={handleTaskDelete}

onTaskStatusChange={handleTaskStatusChange}

onOpenFilters={() => setShowFilters(true)}

onInlineUpdate={handleInlineUpdate}

onTasksReorder={handleTasksReorder}

assignees={assignees}

/>

{/\* Advanced Filters \*/}

<AdvancedFilters

isOpen={showFilters}

onClose={() => setShowFilters(false)}

filters={{}}

onFiltersChange={handleFiltersChange}

onSaveFilter={handleSaveFilter}

assignees={assignees}

clients={clients}

groups={groups}

projects={projects}

/>

{/\* Toast Notifications \*/}

{toast && (

<Toast

message={toast.message}

type={toast.type}

onClose={() => setToast(null)}

/>

)}

{/\* CSS Animations \*/}

<style>{`

@keyframes slideUp {

from {

transform: translateY(100%);

opacity: 0;

}

to {

transform: translateY(0);

opacity: 1;

}

}

@keyframes slideDown {

from {

opacity: 0;

transform: translateY(-10px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

@keyframes fadeIn {

from {

opacity: 0;

}

to {

opacity: 1;

}

}

.animate-slideUp {

animation: slideUp 0.3s ease-out;

}

.animate-slideDown {

animation: slideDown 0.2s ease-out;

}

.animate-fadeIn {

animation: fadeIn 0.2s ease-out;

}

/\* Stili aggiuntivi per drag & drop \*/

.dragging {

opacity: 0.5;

cursor: grabbing !important;

}

.drag-over {

border-top: 3px solid #3b82f6;

}

.task-row-transition {

transition: transform 0.2s ease-in-out;

}

.grip-handle:hover {

color: #6b7280;

transform: scale(1.1);

}

`}</style>

</div>

);

}