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

import { useForm } from "react-hook-form";

import { zodResolver } from "@hookform/resolvers/zod";

import { useMutation, useQuery } from "@tanstack/react-query";

import { z } from "zod";

import { apiRequest, queryClient } from "@/lib/queryClient";

import { useToast } from "@/hooks/use-toast";

import { Button } from "@/components/ui/button";

import { Input } from "@/components/ui/input";

import { Textarea } from "@/components/ui/textarea";

import { Switch } from "@/components/ui/switch";

import { Checkbox } from "@/components/ui/checkbox";

import { Label } from "@/components/ui/label";

import {

Form,

FormControl,

FormDescription,

FormField,

FormItem,

FormLabel,

FormMessage,

} from "@/components/ui/form";

import {

Select,

SelectContent,

SelectGroup,

SelectItem,

SelectLabel,

SelectTrigger,

SelectValue,

} from "@/components/ui/select";

import {

Accordion,

AccordionContent,

AccordionItem,

AccordionTrigger,

} from "@/components/ui/accordion";

import { Badge } from "@/components/ui/badge";

import { BookingPage as BookingPageSchema, insertBookingPageSchema } from "@shared/schema";

import { Loader2, Search } from "lucide-react";

// Extend the schema for frontend validation

const bookingPageFormSchema = insertBookingPageSchema.extend({

name: z.string().min(3, "Name must be at least 3 characters"),

slug: z.string().min(3, "Slug must be at least 3 characters")

.regex(/^[a-z0-9-]+$/, "Slug can only contain lowercase letters, numbers, and hyphens"),

title: z.string().min(3, "Title must be at least 3 characters"),

});

// Type definitions for facilities and appointment types

type Facility = {

id: number;

name: string;

address1: string;

city: string;

state: string;

[key: string]: any;

};

type AppointmentType = {

id: number;

name: string;

description: string;

facilityId: number;

duration: number;

color: string;

type: string;

[key: string]: any;

};

// Full booking page type definition - extend the schema one

type BookingPageExtended = {

id: number;

name: string;

slug: string;

title: string;

description?: string;

welcomeMessage?: string;

confirmationMessage?: string;

isActive: boolean;

facilities: number[] | Record<string, any>; // Can be array or object

excludedAppointmentTypes?: number[];

useOrganizationLogo: boolean;

customLogo?: string | null;

primaryColor?: string;

createdBy: number;

createdAt: string | Date;

lastModifiedAt?: string | Date | null;

lastModifiedBy?: number | null;

};

// Create separate components to isolate state updates

interface FacilityItemProps {

facility: Facility;

isSelected: boolean;

onToggle: (id: number, checked: boolean) => void;

selectedTypes?: number[];

facilityTypes?: number;

}

// Isolated FacilityItem with its own state management to prevent re-render loops

function FacilityItem({ facility, isSelected: initialIsSelected, onToggle, selectedTypes = [], facilityTypes = 0 }: FacilityItemProps) {

// Local component state to prevent propagation of render cycles

const [isSelected, setIsSelected] = useState(initialIsSelected);

// Effect to sync with parent state when it changes externally

useEffect(() => {

if (isSelected !== initialIsSelected) {

setIsSelected(initialIsSelected);

}

}, [initialIsSelected]);

// Count of selected types for this facility

const selectedCount = selectedTypes.length;

// Local handler that updates local state first to avoid cascading updates

const handleToggle = () => {

const newState = !isSelected;

setIsSelected(newState);

// Then notify parent

onToggle(facility.id, newState);

};

return (

<div

className={`flex items-start space-x-2 p-3 border rounded-md hover:bg-muted/30 transition-colors cursor-pointer ${

isSelected ? 'border-primary/50 bg-primary/5' : ''

}`}

onClick={handleToggle}

>

<div className="mt-1">

<input

type="checkbox"

id={`facility-${facility.id}`}

checked={isSelected}

onChange={handleToggle}

className="h-4 w-4 rounded border-gray-300 text-primary focus:ring-primary"

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

/>

</div>

<div className="grid gap-1 leading-none flex-grow">

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

<Label

htmlFor={`facility-${facility.id}`}

className="text-sm font-medium leading-none cursor-pointer"

>

{facility.name}

</Label>

<Badge variant={selectedCount > 0 ? "default" : "outline"} className="ml-2">

{selectedCount}/{facilityTypes} Types

</Badge>

</div>

<p className="text-xs text-muted-foreground">

{facility.address1}, {facility.city}, {facility.state}

</p>

</div>

</div>

);

}

interface AppointmentTypeItemProps {

type: AppointmentType;

isSelected: boolean;

onToggle: (facilityId: number, typeId: number, checked: boolean) => void;

}

// Completely rebuilt appointment type item with its own local state

// This isolates state management to prevent recursive rendering issues

function AppointmentTypeItem({ type, isSelected: initialIsSelected, onToggle }: AppointmentTypeItemProps) {

// Internal component state that mirrors the parent state

const [isSelected, setIsSelected] = useState(initialIsSelected);

// Effect to sync with parent state when it changes externally

useEffect(() => {

if (isSelected !== initialIsSelected) {

setIsSelected(initialIsSelected);

}

}, [initialIsSelected]);

// Local handler that updates local state first

const handleToggle = () => {

const newState = !isSelected;

setIsSelected(newState);

// Then propagate the change to parent

onToggle(type.facilityId, type.id, newState);

};

// Updated UI to avoid any potential circular updates

return (

<div

className={`px-3 py-2 flex items-start space-x-2 hover:bg-muted/20 transition-colors cursor-pointer ${

isSelected ? 'bg-primary/5' : ''

}`}

>

<div className="mt-1">

<input

type="checkbox"

id={`type-${type.facilityId}-${type.id}`}

checked={isSelected}

onChange={handleToggle}

className="h-4 w-4 rounded border-gray-300 text-primary focus:ring-primary"

/>

</div>

<div

onClick={handleToggle}

className="flex-grow"

>

<Label

htmlFor={`type-${type.facilityId}-${type.id}`}

className="text-sm font-medium cursor-pointer"

>

{type.name}

</Label>

<div className="flex items-center mt-1">

<div

className="w-3 h-3 rounded-full mr-2"

style={{ backgroundColor: type.color }}

/>

<span className="text-xs text-muted-foreground">

{type.duration} minutes

</span>

</div>

</div>

</div>

);

}

type BookingPageFormProps = {

bookingPage?: BookingPageSchema;

onSuccess: () => void;

onCancel: () => void;

};

export default function BookingPageForm({ bookingPage, onSuccess, onCancel }: BookingPageFormProps) {

// Make the form dialog scrollable with a max height

useEffect(() => {

// Add scrollable class to the closest dialog container

const dialogContent = document.querySelector('.booking-page-form-dialog-content');

if (dialogContent) {

dialogContent.classList.add('max-h-[80vh]', 'overflow-y-auto');

}

}, []);

const { toast } = useToast();

const [selectedFacilities, setSelectedFacilities] = useState<number[]>([]);

const [selectedAppointmentTypes, setSelectedAppointmentTypes] = useState<Record<number, number[]>>({});

const [searchTerm, setSearchTerm] = useState<string>("");

const [openAccordionItems, setOpenAccordionItems] = useState<string[]>([]);

// Queries for data

const { data: facilitiesData = [], isLoading: isLoadingFacilities } = useQuery<Facility[]>({

queryKey: ['/api/facilities'],

retry: false

});

// Get appointment types query for each facility

const { data: appointmentTypesData, isLoading: isLoadingAppointmentTypes } = useQuery<Record<string, AppointmentType>>({

queryKey: ['/api/appointment-types'],

enabled: !!facilitiesData,

retry: false

});

// Type safe references to the data

const facilities = facilitiesData;

const appointmentTypes = appointmentTypesData || {};

// Initialize form with default values or existing booking page

const form = useForm<z.infer<typeof bookingPageFormSchema>>({

resolver: zodResolver(bookingPageFormSchema),

defaultValues: bookingPage

? {

name: bookingPage.name,

slug: bookingPage.slug,

title: bookingPage.title,

description: bookingPage.description,

welcomeMessage: bookingPage.welcomeMessage,

confirmationMessage: bookingPage.confirmationMessage,

customLogo: bookingPage.customLogo,

useOrganizationLogo: bookingPage.useOrganizationLogo,

primaryColor: bookingPage.primaryColor || "#22c55e",

isActive: bookingPage.isActive

}

: {

name: "",

slug: "",

title: "",

description: "",

welcomeMessage: "Book your appointment online",

confirmationMessage: "Your appointment has been successfully booked.",

customLogo: "",

useOrganizationLogo: true,

primaryColor: "#22c55e",

isActive: true

}

});

// Initialize selected facilities and appointment types if editing

useEffect(() => {

if (bookingPage && bookingPage.facilities) {

// Type checking for facilities

let facilityIds: number[] = [];

// Handle different formats of facilities data

if (Array.isArray(bookingPage.facilities)) {

// Direct array of facility IDs

facilityIds = bookingPage.facilities as number[];

setSelectedFacilities(facilityIds);

} else if (typeof bookingPage.facilities === 'object' && bookingPage.facilities !== null) {

// Legacy format - object with facility IDs as keys

facilityIds = Object.keys(bookingPage.facilities).map(id => Number(id));

setSelectedFacilities(facilityIds);

}

// If we have facilities and appointment types data, set up the appointment types

if (facilityIds.length > 0 && facilities && appointmentTypes) {

const typesMap: Record<number, number[]> = {};

// Initialize with empty arrays for each selected facility

facilityIds.forEach(facilityId => {

typesMap[facilityId] = [];

});

// Get all excluded appointment types

const excludedTypes = bookingPage.excludedAppointmentTypes

? (bookingPage.excludedAppointmentTypes as number[])

: [];

// For each facility, add all appointment types for that facility that are not excluded

Object.values(appointmentTypes).forEach(type => {

const facilityId = type.facilityId;

// Only process if this facility is selected

if (facilityIds.includes(facilityId)) {

// Add the appointment type if it's not excluded

if (!excludedTypes.includes(type.id)) {

typesMap[facilityId] = [...(typesMap[facilityId] || []), type.id];

}

}

});

setSelectedAppointmentTypes(typesMap);

}

// Set initial open accordion items for facilities with selections

const openItems = facilityIds.map(id => `facility-${id}`);

setOpenAccordionItems(openItems);

}

}, [bookingPage, facilities, appointmentTypes]);

// Debounce search input to prevent too many re-renders

const [debouncedSearchTerm, setDebouncedSearchTerm] = useState(searchTerm);

useEffect(() => {

const timer = setTimeout(() => {

setDebouncedSearchTerm(searchTerm);

}, 300);

return () => clearTimeout(timer);

}, [searchTerm]);

// Filtered appointment types based on search term

const filteredAppointmentTypes = useMemo(() => {

if (!appointmentTypesData) return {} as Record<string, AppointmentType>;

if (!debouncedSearchTerm.trim()) return appointmentTypesData;

return Object.entries(appointmentTypesData).reduce<Record<string, AppointmentType>>(

(filtered, [typeId, appointmentType]) => {

if (appointmentType.name.toLowerCase().includes(debouncedSearchTerm.toLowerCase())) {

filtered[typeId] = appointmentType;

}

return filtered;

}, {});

}, [appointmentTypesData, debouncedSearchTerm]);

// Completely redesigned appointment type toggle to eliminate any possible render loops

const toggleAppointmentType = useCallback((facilityId: number, appointmentTypeId: number, checked: boolean) => {

console.log(`Toggling appointment type ${appointmentTypeId} for facility ${facilityId} to ${checked ? 'selected' : 'unselected'}`);

setSelectedAppointmentTypes(currentState => {

// Create a brand new state object (immutable)

const newState = {...currentState};

// Initialize array for this facility if it doesn't exist

if (!newState[facilityId]) {

newState[facilityId] = [];

}

if (checked) {

// Only add if not already selected

if (!newState[facilityId].includes(appointmentTypeId)) {

newState[facilityId] = [...newState[facilityId], appointmentTypeId];

console.log(`Added type ${appointmentTypeId} to facility ${facilityId}`);

}

} else {

// Only remove if currently selected

if (newState[facilityId].includes(appointmentTypeId)) {

newState[facilityId] = newState[facilityId].filter(id => id !== appointmentTypeId);

console.log(`Removed type ${appointmentTypeId} from facility ${facilityId}`);

}

}

return newState;

});

}, []); // Empty dependency array - this function never needs to be recreated

// Create mutation with improved error handling and consistent payload structure

const createMutation = useMutation({

mutationFn: async (data: z.infer<typeof bookingPageFormSchema>) => {

// Get all selected appointment type IDs from selectedAppointmentTypes

const appointmentTypes: number[] = [];

// Gather all selected appointment types across facilities

Object.entries(selectedAppointmentTypes).forEach(([facilityId, typeIds]) => {

appointmentTypes.push(...typeIds);

});

// Create the payload with the exact structure expected by the API

const payload = {

...data,

facilities: selectedFacilities,

appointmentTypes: appointmentTypes

};

console.log("Creating booking page with payload:", payload);

try {

const response = await apiRequest('POST', '/api/booking-pages', payload);

if (!response.ok) {

// Try to get the error message from the response

let errorMessage = "Failed to create booking page";

try {

const errorJson = await response.json();

errorMessage = errorJson.message || errorJson.error || errorMessage;

} catch {

// If we can't parse the JSON, try text

const errorText = await response.text();

errorMessage = errorText || errorMessage;

}

console.error("API error:", response.status, errorMessage);

throw new Error(errorMessage);

}

const responseData = await response.json();

console.log("Create successful, received response:", responseData);

return responseData;

} catch (error) {

console.error("Error in createMutation:", error);

throw error;

}

},

onSuccess: () => {

toast({

title: "Success",

description: "Booking page created successfully",

variant: "default",

});

queryClient.invalidateQueries({ queryKey: ['/api/booking-pages'] });

onSuccess();

},

onError: (error) => {

console.error("Error creating booking page:", error);

toast({

title: "Error",

description: error instanceof Error ? error.message : "Failed to create booking page",

variant: "destructive",

});

}

});

// Update mutation with enhanced reliability and fixed payload structure

const updateMutation = useMutation({

mutationFn: async (data: z.infer<typeof bookingPageFormSchema>) => {

if (!bookingPage) {

console.error("No booking page provided to update");

throw new Error("Cannot update non-existent booking page");

}

console.log("Starting booking page update process with ID:", bookingPage.id);

// Get all selected appointment type IDs from selectedAppointmentTypes

const appointmentTypes: number[] = [];

// Gather all selected appointment types across facilities

Object.entries(selectedAppointmentTypes).forEach(([facilityId, typeIds]) => {

appointmentTypes.push(...typeIds);

});

// Create the payload with the exact structure expected by the API

const payload = {

...data,

facilities: selectedFacilities,

appointmentTypes: appointmentTypes

};

console.log("Sending update API request with payload:", payload);

try {

const response = await apiRequest('PUT', `/api/booking-pages/${bookingPage.id}`, payload);

if (!response.ok) {

// Try to get the error message from the response

let errorMessage = "Failed to update booking page";

try {

const errorJson = await response.json();

errorMessage = errorJson.message || errorJson.error || errorMessage;

} catch {

// If we can't parse the JSON, try text

const errorText = await response.text();

errorMessage = errorText || errorMessage;

}

console.error("API error:", response.status, errorMessage);

throw new Error(errorMessage);

}

const data = await response.json();

console.log("Update successful, received response:", data);

return data;

} catch (error) {

console.error("Error in updateMutation:", error);

throw error;

}

},

onSuccess: (data) => {

console.log("Update mutation succeeded with data:", data);

toast({

title: "Success",

description: "Booking page updated successfully",

variant: "default",

});

queryClient.invalidateQueries({ queryKey: ['/api/booking-pages'] });

onSuccess();

},

onError: (error) => {

console.error("Error updating booking page:", error);

toast({

title: "Error",

description: error instanceof Error ? error.message : "Failed to update booking page. Please try again.",

variant: "destructive",

});

}

});

// Handle form submission with enhanced debugging and improved validation

const onSubmit = (data: z.infer<typeof bookingPageFormSchema>) => {

console.log("Form submission triggered with data:", data);

console.log("Selected facilities:", selectedFacilities);

console.log("Selected appointment types:", selectedAppointmentTypes);

// Check if any facilities are selected

if (selectedFacilities.length === 0) {

console.log("Error: No facilities selected");

toast({

title: "Error",

description: "Please select at least one facility",

variant: "destructive",

});

return;

}

// Check if at least one appointment type is selected across all facilities

const hasAnyTypeSelected = Object.values(selectedAppointmentTypes).some(types => types.length > 0);

if (!hasAnyTypeSelected) {

console.log("Error: No appointment types selected");

toast({

title: "Error",

description: "Please select at least one appointment type",

variant: "destructive",

});

return;

}

try {

if (bookingPage) {

console.log("Updating existing booking page with ID:", bookingPage.id);

updateMutation.mutate(data);

} else {

console.log("Creating new booking page");

createMutation.mutate(data);

}

} catch (error) {

console.error("Error in form submission:", error);

toast({

title: "Error",

description: "An unexpected error occurred while saving the booking page",

variant: "destructive",

});

}

};

// Toggle facility selection with immediate accordion expansion - using useCallback

const toggleFacility = useCallback((facilityId: number, checked: boolean) => {

console.log(`Toggling facility ${facilityId} to ${checked ? 'selected' : 'unselected'}`);

// Update facilities state

setSelectedFacilities(prevFacilities => {

// No change needed if already in the desired state

if (checked && prevFacilities.includes(facilityId)) return prevFacilities;

if (!checked && !prevFacilities.includes(facilityId)) return prevFacilities;

// Either add or remove the facility

if (checked) {

return [...prevFacilities, facilityId];

} else {

return prevFacilities.filter(id => id !== facilityId);

}

});

// Update appointment types

setSelectedAppointmentTypes(prevTypes => {

const newTypes = {...prevTypes};

if (checked) {

// Get all appointment types for this facility

// Using closure over appointmentTypes, avoiding it as a dependency

const facilityAppointmentTypes = Object.values(appointmentTypes)

.filter(type => type.facilityId === facilityId)

.map(type => type.id);

// Add all types for this facility

if (facilityAppointmentTypes.length > 0) {

newTypes[facilityId] = facilityAppointmentTypes;

}

} else {

// Remove appointment types for this facility

delete newTypes[facilityId];

}

return newTypes;

});

// Update accordion state

setOpenAccordionItems(prevItems => {

const itemKey = `facility-${facilityId}`;

if (checked) {

// Only add if not already included

if (!prevItems.includes(itemKey)) {

return [...prevItems, itemKey];

}

} else {

// Only remove if currently included

if (prevItems.includes(itemKey)) {

return prevItems.filter(item => item !== itemKey);

}

}

return prevItems; // No change needed

});

}, []); // Empty dependency array - relies on closures

const isPending = createMutation.isPending || updateMutation.isPending;

const isLoading = isLoadingFacilities || isLoadingAppointmentTypes;

// Helper to slugify the name

const generateSlug = (name: string) => {

return name

.toLowerCase()

.replace(/[^\w\s-]/g, '')

.replace(/[\s\_-]+/g, '-')

.replace(/^-+|-+$/g, '');

};

return (

<Form {...form}>

<form onSubmit={form.handleSubmit(onSubmit)} className="space-y-6 booking-page-form-dialog-content max-h-[80vh] overflow-y-auto pr-1">

{/\* Basic Information \*/}

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<FormField

control={form.control}

name="name"

render={({ field }) => (

<FormItem>

<FormLabel>Name</FormLabel>

<FormControl>

<Input

placeholder="Enter booking page name"

{...field}

onChange={(e) => {

field.onChange(e);

// If this is a new booking page and slug is empty, generate a slug

if (!bookingPage && !form.getValues("slug")) {

form.setValue("slug", generateSlug(e.target.value));

}

}}

/>

</FormControl>

<FormDescription>

Internal name for this booking page.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

<FormField

control={form.control}

name="slug"

render={({ field }) => (

<FormItem>

<FormLabel>Slug</FormLabel>

<FormControl>

<Input

placeholder="booking-page-url"

{...field}

/>

</FormControl>

<FormDescription>

URL-friendly identifier. Used in the page URL.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

</div>

<FormField

control={form.control}

name="title"

render={({ field }) => (

<FormItem>

<FormLabel>Page Title</FormLabel>

<FormControl>

<Input

placeholder="Enter page title"

{...field}

/>

</FormControl>

<FormDescription>

Title displayed to users on the booking page.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

<FormField

control={form.control}

name="description"

render={({ field }) => (

<FormItem>

<FormLabel>Description</FormLabel>

<FormControl>

<Textarea

placeholder="Enter page description"

{...field}

value={field.value || ""}

/>

</FormControl>

<FormDescription>

Brief description of the booking page.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<FormField

control={form.control}

name="welcomeMessage"

render={({ field }) => (

<FormItem>

<FormLabel>Introduction Text</FormLabel>

<FormControl>

<Textarea

placeholder="Welcome message for visitors"

{...field}

value={field.value || ""}

/>

</FormControl>

<FormDescription>

Introduction text displayed at the top of the booking page.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

<FormField

control={form.control}

name="confirmationMessage"

render={({ field }) => (

<FormItem>

<FormLabel>Success Message</FormLabel>

<FormControl>

<Textarea

placeholder="Message shown after successful booking"

{...field}

value={field.value || ""}

/>

</FormControl>

<FormDescription>

Message displayed after a successful booking.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

</div>

<FormField

control={form.control}

name="useOrganizationLogo"

render={({ field }) => (

<FormItem className="flex flex-row items-center justify-between rounded-lg border p-4">

<div className="space-y-0.5">

<FormLabel className="text-base">Use Organization Logo</FormLabel>

<FormDescription>

Use the logo from organization settings.

</FormDescription>

</div>

<FormControl>

<Switch

checked={field.value}

onCheckedChange={field.onChange}

/>

</FormControl>

</FormItem>

)}

/>

<div className="grid grid-cols-1 md:grid-cols-2 gap-6">

<FormField

control={form.control}

name="customLogo"

render={({ field }) => (

<FormItem>

<FormLabel>Custom Logo URL</FormLabel>

<FormControl>

<Input

placeholder="https://example.com/logo.png"

{...field}

value={field.value || ""}

disabled={form.getValues("useOrganizationLogo")}

/>

</FormControl>

<FormDescription>

URL to a custom logo if not using organization logo.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

<FormField

control={form.control}

name="primaryColor"

render={({ field }) => (

<FormItem>

<FormLabel>Primary Color</FormLabel>

<div className="flex gap-2">

<FormControl>

<Input

type="color"

{...field}

value={field.value || "#22c55e"}

className="w-12 h-9 p-1"

/>

</FormControl>

<FormControl>

<Input

type="text"

{...field}

value={field.value || "#22c55e"}

className="flex-1"

/>

</FormControl>

</div>

<FormDescription>

Primary color for the booking page theme.

</FormDescription>

<FormMessage />

</FormItem>

)}

/>

</div>

<FormField

control={form.control}

name="isActive"

render={({ field }) => (

<FormItem className="flex flex-row items-center justify-between rounded-lg border p-4">

<div className="space-y-0.5">

<FormLabel className="text-base">Active Status</FormLabel>

<FormDescription>

Enable or disable this booking page.

</FormDescription>

</div>

<FormControl>

<Switch

checked={field.value}

onCheckedChange={field.onChange}

/>

</FormControl>

</FormItem>

)}

/>

{/\* Facilities & Appointment Types Section \*/}

<div className="border rounded-lg p-5 shadow-sm">

<h3 className="text-lg font-medium mb-4">Facilities & Appointment Types</h3>

{/\* Search box for appointment types \*/}

<div className="relative mb-4">

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

<Input

placeholder="Search facilities or appointment types..."

className="pl-9"

value={searchTerm}

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

/>

</div>

{/\* Facilities section \*/}

<div className="mb-6">

<div className="flex items-center justify-between mb-3">

<h4 className="font-medium text-base">Select Facilities</h4>

{selectedFacilities.length > 0 && (

<Badge variant="outline" className="bg-primary/10 text-primary">

{selectedFacilities.length} selected

</Badge>

)}

</div>

{isLoading ? (

<div className="flex justify-center py-4">

<Loader2 className="h-6 w-6 animate-spin text-primary" />

</div>

) : (

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

{facilities && facilities.length > 0 ? (

facilities

.filter(facility =>

!debouncedSearchTerm ||

facility.name.toLowerCase().includes(debouncedSearchTerm.toLowerCase()) ||

facility.city.toLowerCase().includes(debouncedSearchTerm.toLowerCase())

)

.map((facility) => {

// Get all appointment types for this facility

const facilityTypes = Object.values(appointmentTypes)

.filter(type => type.facilityId === facility.id)

.length;

// Get the selected appointment types for this facility

const selectedTypes = selectedAppointmentTypes[facility.id] || [];

return (

<FacilityItem

key={facility.id}

facility={facility}

isSelected={selectedFacilities.includes(facility.id)}

onToggle={toggleFacility}

selectedTypes={selectedTypes}

facilityTypes={facilityTypes}

/>

);

})

) : (

<div className="text-center py-4 text-muted-foreground col-span-2">

No facilities found. Please create facilities first.

</div>

)}

</div>

)}

</div>

{/\* Appointment Types Section \*/}

{selectedFacilities.length > 0 && (

<div>

<h4 className="font-medium text-base mb-3">Appointment Types</h4>

<div className="max-h-[300px] overflow-y-auto pr-1 space-y-3">

{facilities

.filter(facility => selectedFacilities.includes(facility.id))

.map(facility => {

// Filter appointment types for this facility

const facilityAppointmentTypes = appointmentTypes

? Object.values(appointmentTypes)

.filter(type => type.facilityId === facility.id)

: [];

// Apply search filter if there's a search term

const filteredTypes = debouncedSearchTerm

? facilityAppointmentTypes.filter(type =>

type.name.toLowerCase().includes(debouncedSearchTerm.toLowerCase()))

: facilityAppointmentTypes;

// Skip rendering if no types match search

if (filteredTypes.length === 0 && debouncedSearchTerm) {

return null;

}

// Count how many are selected

const selectedCount = selectedAppointmentTypes[facility.id]?.length || 0;

return (

<Accordion

key={`facility-types-${facility.id}`}

type="multiple"

value={openAccordionItems}

onValueChange={setOpenAccordionItems}

className="rounded-md border"

>

<AccordionItem value={`facility-${facility.id}`} className="border-none">

<div className="px-3 py-2 flex justify-between items-center bg-muted/20">

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

<AccordionTrigger className="py-0 hover:no-underline">

<span className="font-medium">{facility.name}</span>

</AccordionTrigger>

</div>

{facilityAppointmentTypes.length > 0 && (

<Badge variant="outline" className="bg-primary/10 text-primary text-xs">

{selectedCount}/{facilityAppointmentTypes.length} Types

</Badge>

)}

</div>

<AccordionContent className="pt-2 pb-0">

{filteredTypes.length > 0 ? (

<div className="divide-y">

{filteredTypes.map(type => (

<AppointmentTypeItem

key={type.id}

type={type}

isSelected={(selectedAppointmentTypes[facility.id] || []).includes(type.id)}

onToggle={toggleAppointmentType}

/>

))}

</div>

) : (

<div className="text-center py-2 px-3 text-muted-foreground">

No appointment types match your search.

</div>

)}

</AccordionContent>

</AccordionItem>

</Accordion>

);

})}

</div>

</div>

)}

</div>

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

<Button type="button" variant="outline" onClick={onCancel}>

Cancel

</Button>

<Button

type="submit" // Changed to submit to trigger the form's onSubmit

disabled={isPending}

>

{isPending ? (

<>

<Loader2 className="mr-2 h-4 w-4 animate-spin" />

{bookingPage ? "Updating..." : "Creating..."}

</>

) : (

<>{bookingPage ? "Update" : "Create"} Booking Page</>

)}

</Button>

</div>

</form>

</Form>

);

}