// HomeContent.tsx

```tsx
import React, { useState } from "react";
import { CustomerForm } from "./CustomerForm";
import { AddressForm } from "./AddressForm";
import Header from "./Header";
import SurveyInformation from "./SurveyInformation";
import SlotList from "./SlotList";
import { DatePicker } from "./DatePicker";
import { Warehouse, driverMapping } from "../data/DriverLookupTable";
import { SecondaryContact } from "./SecondaryContact";
import { TenantContact } from "./TenantContact";
import { ProjectQuestions } from "./ProjectQuestions";
import RecommendedTimeSlots from './RecommendedTimeSlots';

interface FormData {
 firstName: string;
 lastName: string;
 language: string;
 streetAddress: string;
 city: string;
 state: string;
 postalCode: string;
 warehouse: Warehouse;
 hasSecondaryContact: boolean;
 secondaryContactName: string;
 secondaryContactRelationship: string;
 secondaryContactPhone: string;
 secondaryContactEmail: string;
 hasTenants: boolean;
 tenantName: string;
 tenantPhone: string;
 tenantEmail: string;
 hasBatteries: boolean;
 designPreference: string;
 isSharedRoof: boolean;
 hasRecentConstruction: boolean;
 hasVaultedCeilings: boolean;
 stories: string;
 hasOngoingConstruction: boolean;
 ongoingConstructionDetails?: string;
 hasExistingSolar: boolean;
```

```
 hasHOA: boolean;
 hasHOAContact: boolean;
 hoaName?: string;
 hoaPhone?: string;
 hoaEmail?: string;
 hasOutsidePanel: boolean;
 hasBlockedAccess: boolean;
 gateAccessType: "yes" | "code" | "no" | "";
 gateCode?: string;
}

interface HomeContentProps {
 formData: FormData;
 setFormData: React.Dispatch<React.SetStateAction<FormData>>;
 availableSlots: any[];
 setAvailableSlots: React.Dispatch<React.SetStateAction<any[]>>;
 selectedSlot: { value: string; reservationId: string } | null;
 setSelectedSlot: React.Dispatch<React.SetStateAction<{ value: string; reservationId:
string } | null>>;
 selectedDate: Date | null;
 setSelectedDate: React.Dispatch<React.SetStateAction<Date | null>>;
 orderNo: string | null;
 setOrderNo: React.Dispatch<React.SetStateAction<string | null>>;
}

const HomeContent: React.FC<HomeContentProps> = ({
 formData,
 setFormData,
 availableSlots,
 setAvailableSlots,
 selectedSlot,
 setSelectedSlot,
 selectedDate,
 setSelectedDate,
 orderNo,
 setOrderNo,
}) => {
 // Handle the form input changes
 const handleChange = (
   e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement | HTMLTextAreaElement>
 ) => {
   const { id, value } = e.target;
```

```
  setFormData((prev) => ({
    ...prev,
    [id]: value,
  }));
};
const [slotErrorMessage, setSlotErrorMessage] = useState(""); // For fetchSlots
errors
const [orderErrorMessage, setOrderErrorMessage] = useState(""); // For updateOrder
errors
const [successMessage, setSuccessMessage] = useState(""); // For success messages
const [loading, setLoading] = useState(false); // For global loading state
const [showSlots, setShowSlots] = useState(false);

// Get drivers based on the warehouse
const drivers = driverMapping[formData.warehouse] || [];

// Function to handle slot selection
const handleSlotSelect = async (value: string, reservationId: string) => {
  try {
    console.log("Selecting slot:", { value, reservationId });

    // Update UI first
    setSelectedSlot({ value, reservationId });

    // Extract times
    const [fromDateTime, toDateTime] = value.split(' - ');
    const fromTime = fromDateTime.split('T')[1].substring(0, 5);
    const toTime = toDateTime.split('T')[1].substring(0, 5);

    // Update existing order with new time window
    const updatePayload = {
      orders: [{
        operation: "UPDATE", // Changed from CREATE to UPDATE
        orderNo: orderNo,
        timeWindows: [{
          twFrom: fromTime,
          twTo: toTime
        }]
      }]
    };
```

```
    console.log("Updating order with payload:", JSON.stringify(updatePayload, null,
2));

    const updateResponse = await fetch("/api/updateOrder", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(updatePayload),
    });

    const updateData = await updateResponse.json();
    console.log("Update order response:", updateData);

  } catch (error) {
    console.error("Error in slot selection:", error);
  }
};

// Function to handle date selection and API call to get available slots
const handleDateSelect = async (date: Date | undefined) => {
  if (!date) return;

  setSelectedDate(date);
  // Format the date as UTC (e.g., 2024-11-20)
  const formattedDate = new Date(
    Date.UTC(date.getFullYear(), date.getMonth(), date.getDate())
  )
    .toISOString()
    .split("T")[0];
  const generatedOrderNo = `BE${Date.now()}`;
  setOrderNo(generatedOrderNo);

  // Get drivers only if warehouse is not Out of Region
  const drivers = formData.warehouse !== "Out of Region" ?
driverMapping[formData.warehouse] || [] : [];

  // Construct API request payload
  const bookingData = {
    order: {
      operation: "CREATE",
      orderNo: generatedOrderNo,
```

```
      type: "D",
      location: {
        locationName: "Project Address",
        address: `${formData.streetAddress}, ${formData.city}, ${formData.state}
${formData.postalCode}, United States`,
      },
      duration: 60, // Duration in minutes
    },
    slots: {
      dates: [formattedDate],
      timeWindows: [
        { twFrom: "08:00", twTo: "10:00" },
        { twFrom: "10:00", twTo: "12:00" },
        { twFrom: "12:00", twTo: "14:00" },
        { twFrom: "14:00", twTo: "16:00" },
        { twFrom: "16:00", twTo: "18:00" },
      ],
    },
    planning: {
      // Only include useDrivers if we have drivers
      ...(drivers.length > 0 && { useDrivers: drivers }),
      clustering: false,
      lockType: "RESOURCES"
    },
  };
  console.log(bookingData);

  try {
    setLoading(true);
    const response = await fetch("/api/fetchSlots", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(bookingData),
    });

    if (response.ok) {
      const responseData = await response.json();
      console.log("Fetched slots:", responseData);

      if (responseData.success) {
```

```
        setAvailableSlots(responseData.slots || []);
        setSlotErrorMessage(""); // Clear any previous errors
      } else {
        console.error("Failed to fetch slots:", responseData.message);
        setAvailableSlots([]);
        setSlotErrorMessage(responseData.message || "Failed to fetch slots.");
      }
    } else {
      console.error("API call failed:", response.statusText);
      setAvailableSlots([]);
      setSlotErrorMessage(response.statusText || "API call failed.");
    }
  } catch (error: any) {
    console.error("Error calling API:", error);
    setAvailableSlots([]);
    setSlotErrorMessage(
      error.message || "An unexpected error occurred while fetching slots."
    );
  } finally {
    setLoading(false); // Ensure loading is reset
  }
};

const handleSubmit = async () => {
  if (!selectedSlot || !selectedSlot.reservationId) {
    setOrderErrorMessage("Please select a time slot before submitting.");
    return;
  }

  if (!orderNo) {
    setOrderErrorMessage("Order number is missing. Please try again.");
    return;
  }

  try {
    setLoading(true);

    // First make the reservation
    const reservationData = {
      payload: {
        reservationId: selectedSlot.reservationId,
        orderNo,
```

```
        surveyDate: selectedDate ?
          new Date(selectedDate.getTime() + (24 * 60 * 60 *
1000)).toISOString().split('T')[0]
          : null,
        timeSlot: selectedSlot.value,
        formData
      }
    };

    // Step 1: Send data to the webhook
    const webhookResponse = await
fetch("https://webhooks.workato.com/webhooks/rest/49f37552-9667-45f9-a324-1b4f58455632
/survey-optimo", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(reservationData),
    });

    if (!webhookResponse.ok) {
      const errorText = await webhookResponse.text();
      console.error("Webhook error response:", errorText);
      setOrderErrorMessage("Failed to send data. Please try again.");
      return;
    }

    // Step 2: Make reservation with OptimoRoute
    const reservationResponse = await fetch("/api/MakeReservation", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        reservationId: selectedSlot.reservationId,
        createOrderIfBookingFails: false,
      }),
    });

    if (!reservationResponse.ok) {
      const errorText = await reservationResponse.text();
      console.error("Reservation error response:", errorText);
```

```
      setOrderErrorMessage("Failed to make reservation. Please try again.");
      return;
    }

    // Wait for 2 seconds to allow order creation to complete
    await new Promise(resolve => setTimeout(resolve, 2000));

    // Then update the time window
    const [fromDateTime, toDateTime] = selectedSlot.value.split(' - ');
    const fromTime = fromDateTime.split('T')[1].substring(0, 5);
    const toTime = toDateTime.split('T')[1].substring(0, 5);

    const updatePayload = {
      orders: [{
        operation: "UPDATE",
        orderNo: orderNo,
        timeWindows: [{
          twFrom: fromTime,
          twTo: toTime
        }]
      }]
    };

    console.log("\n=== UPDATE PAYLOAD ===");
    console.log(JSON.stringify(updatePayload, null, 2));

    const updateResponse = await fetch("/api/updateOrder", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(updatePayload),
    });

    if (!updateResponse.ok) {
      throw new Error("Failed to update order");
    }

    // If all steps are successful, redirect to the success page
    console.log("Data sent and reservation made successfully");
    const bookingRecap = {
      formData,
      selectedSlot,
      selectedDate,
```

```
      orderNo
    };

    localStorage.setItem('bookingRecap', JSON.stringify(bookingRecap));
    window.location.href = "/submissionSuccessful";
  } catch (error) {
    setOrderErrorMessage("Something went wrong. Please try again later.");
    console.error("Error during submission:", error);
  } finally {
    setLoading(false);
  }
};

const handleValidAddress = () => {
  setShowSlots(true);
};

return (
  <div className="h-full">
    <div className="Form">
      <CustomerForm formData={formData} handleChange={handleChange} />
      <AddressForm
        formData={formData}
        handleChange={handleChange}
        onValidAddress={handleValidAddress}
      />

      {formData.warehouse && (
        <>
          <div className="mt-6 p-4 bg-white dark:bg-gray-800 rounded-lg shadow">
            {formData.warehouse === 'Out of Region' ? (
              <div className="flex items-center space-x-3 mb-4">
                <div className="bg-red-100 dark:bg-red-900 p-2 rounded-full">
                  <svg className="w-6 h-6 text-red-600 dark:text-red-400" fill="none"
stroke="currentColor" viewBox="0 0 24 24">
                    <path strokeLinecap="round" strokeLinejoin="round"
strokeWidth={2} d="M12 9v2m0 4h.01m-6.938 4h13.856c1.54 0 2.502-1.667 1.732-3L13.732
4c-.77-1.333-2.694-1.333-3.464 0L3.34 16c-.77 1.333.192 3 1.732 3z" />
                  </svg>
                </div>
                <div>
```

```
                  <h3 className="text-lg font-semibold text-gray-900
dark:text-white">
                    Outside Service Area
                  </h3>
                  <p className="mt-2 text-gray-600 dark:text-gray-300">
                    We apologize, but this location is outside our current service
area. Please contact Better Earth directly at (888) 444-5555 to discuss survey
options.
                    And checkout our territory map{' '}
                    <a

href="https://help.betterearth.io/territory-map#block-a1ded0c0121942529f2b0973c7a708e6
"
                      target="_blank"
                      rel="noopener noreferrer"
                      className="text-green-600 hover:text-green-700
dark:text-green-400 dark:hover:text-green-300 underline font-medium transition-colors
duration-200"
                    >
                      here
                    </a>.
                  </p>
                </div>
              </div>
            ) : (
              <>

              </>
            )}
          </div>

          {formData.warehouse !== 'Out of Region' && (
            <>
              <RecommendedTimeSlots onPickAnotherDate={() => setShowSlots(true)} />

              {showSlots && (
                <div className="mt-6">
                  <hr className="my-6 h-px bg-gradient-to-r from-green-500/50
to-emerald-600/50 border-0 dark:from-green-500/30 dark:to-emerald-600/30" />
                  <div className="w-full">
                    <h2 className="text-2xl text-center font-semibold text-gray-900
dark:text-white mb-6">
```

```
                      Select Your Preferred Time
                    </h2>

                    <div className="bg-gray-50/50 dark:bg-gray-800/50 rounded-xl p-6
backdrop-blur-sm mb-8">
                      <p className="text-gray-700 dark:text-gray-200">
                        Please select any available slot. This reserves your
preferred day - you'll receive a specific 2-hour arrival window the evening before via
text and email, along with live tracking.
                      </p>

                      <div className="flex items-start space-x-2 mt-4 text-green-600
dark:text-gray-300">
                        <svg className="w-4 h-4 mt-1 flex-shrink-0" fill="none"
stroke="currentColor" viewBox="0 0 24 24">
                          <path strokeLinecap="round" strokeLinejoin="round"
strokeWidth={2} d="M13 16h-1v-4h-1m1-4h.01M21 12a9 9 0 11-18 0 9 9 0 0118 0z" />
                        </svg>
                        <p>
                          Someone must be present during the survey to provide access
to the attic, electrical equipment, and other areas of the property.
                        </p>
                      </div>
                    </div>
                  </div>
                  <div className="grid sm:grid-cols-2">
                    <div className="pt-10 col-span-1 grid place-items-center h-full">
                      <DatePicker onSelect={handleDateSelect} />
                    </div>
                    <div className="slots-container mt-6 col-span-1">
                      <SlotList
                        availableSlots={availableSlots}
                        selectedSlot={selectedSlot}
                        handleSlotSelect={handleSlotSelect}
                      />
                    </div>
                  </div>

                  {/* Display error messages or success messages */}
                  <div className="mt-4 text-center">
                    {slotErrorMessage && (
```

```
                      <p className="text-red-600 font-semibold flex items-center
justify-center">
                        ⚠️ {slotErrorMessage}
                      </p>
                    )}
                    {orderErrorMessage && (
                      <p className="text-red-600 font-semibold flex items-center
justify-center">
                        ⚠️ {orderErrorMessage}
                      </p>
                    )}
                    {successMessage && (
                      <p className="text-green-600 font-semibold flex items-center
justify-center">
                        ✅ {successMessage}
                      </p>
                    )}
                    {loading && (
                      <div className="fixed inset-0 bg-black/20 backdrop-blur-sm flex
items-center justify-center z-50">
                        <div className="bg-white dark:bg-gray-800 rounded-lg p-6
shadow-xl flex flex-col items-center space-y-4">
                          <div className="relative w-12 h-12">
                            <div className="absolute inset-0 border-4
border-green-200 rounded-full"></div>
                            <div className="absolute inset-0 border-4
border-green-500 rounded-full animate-spin border-t-transparent"></div>
                          </div>
                          <p className="text-gray-700 dark:text-gray-200 font-medium
text-lg">
                            Checking availability...
                          </p>
                        </div>
                      </div>
                    )}
                  </div>


                  <hr className="my-6 h-px bg-gradient-to-r from-green-500/50
to-emerald-600/50 border-0 dark:from-green-500/30 dark:to-emerald-600/30" />
```

```
                  <SecondaryContact formData={formData} handleChange={handleChange}
/>

                  <hr className="my-6 h-px bg-gradient-to-r from-green-500/50
to-emerald-600/50 border-0 dark:from-green-500/30 dark:to-emerald-600/30" />

                  <TenantContact formData={formData} handleChange={handleChange} />

                  <hr className="my-6 h-px bg-gradient-to-r from-green-500/50
to-emerald-600/50 border-0 dark:from-green-500/30 dark:to-emerald-600/30" />

                  <ProjectQuestions formData={formData} handleChange={handleChange}
/>



                  <div className="mt-8">
                    <div className="mt-6 flex justify-center items-center">
                      <button
                        onClick={handleSubmit}
                        className="px-8 py-4 text-base font-semibold text-white
                                   bg-gradient-to-r from-green-500 to-emerald-600
                                   hover:from-green-600 hover:to-emerald-700
                                   rounded-lg shadow-lg
                                   transform transition-all duration-200 ease-in-out
hover:scale-[1.02]
                                   focus:outline-none focus:ring-2 focus:ring-offset-2
focus:ring-green-500
                                   disabled:opacity-50 disabled:cursor-not-allowed
                                   dark:focus:ring-offset-gray-800"
                      >
                        Submit
                      </button>
                    </div>
                  </div>
                )}
              </>
            )}
          </>
        )}
      </div>
```

```
   </div>
 );
};

export default HomeContent;
```

// Slot.tsx

```
import * as React from "react";

interface SlotProps {
 valueSlot: string;
 onSelect: (value: string, reservationId: string) => void;
 isSelected: boolean;
 reservationId: string;
}

export function Slot({ valueSlot, onSelect, isSelected, reservationId }: SlotProps) {
 return (
   <button
     onClick={() => onSelect(valueSlot, reservationId)}
     className={`max-w-[200px] mx-auto p-3 rounded-xl transition-all duration-200
ease-in-out
               text-base font-medium block w-full
               ${
                 isSelected
                   ? 'bg-white border-2 border-indigo-500 text-indigo-600'
                   : 'border border-indigo-200 text-indigo-600 hover:border-indigo-400
hover:bg-white'
               }
               shadow-sm hover:shadow-md`}
   >
     {valueSlot}
   </button>
 );
}
```

Slot.tsx

```tsx
// SlotList.tsx
import React from 'react';
import { Slot } from './Slot';

interface SlotListProps {
 availableSlots: any[] | undefined;
 selectedSlot: { value: string; reservationId: string } | null;
 handleSlotSelect: (value: string, reservationId: string) => void;
}

export const SlotList: React.FC<SlotListProps> = ({
 availableSlots = [],
 selectedSlot,
 handleSlotSelect,
}) => {
 if (!availableSlots) return null;

 const formatTime = (time: string) => {
   const timeOnly = time.split('T')[1].substring(0, 5);
   const [hours, minutes] = timeOnly.split(':');
   const hour = parseInt(hours);
   const ampm = hour >= 12 ? 'PM' : 'AM';
   const hour12 = hour % 12 || 12;
   const formattedMinutes = minutes.padStart(2, '0');
   return `${hour12}:${formattedMinutes} ${ampm}`;
 };

 return (
   <div className="space-y-3">
     {availableSlots.length > 0 ? (
       availableSlots.map((slot, index) => {
         const originalSlot = `${slot.from} - ${slot.to}`;
         const formattedSlot = `${formatTime(slot.from)} - ${formatTime(slot.to)}`;
         return (
           <Slot
             key={index}
             valueSlot={formattedSlot}
             onSelect={() => handleSlotSelect(originalSlot, slot.reservationId)}
             isSelected={selectedSlot?.value === originalSlot}
             reservationId={slot.reservationId}
           />
```

```
        );
      })
    ) : (
      <p className="text-gray-500 text-center">No available slots</p>
    )}
  </div>
 );
};

export default SlotList;
```

// fetch slots

```
import { NextResponse } from 'next/server';

interface Slot {
 cost?: number;
 [key: string]: any;
}

export async function POST(request: Request) {
 try {
  const body = await request.json();
  const { order, slots, planning } = body;

  // Construct the API request body for the OptimoRoute API
  const bookingData = {
    order,
    slots,
    planning: {
      ...planning,
      balancing: "ON",  // Balance routes
      balanceBy: "WT",  // Balance by working time
      clustering: false, // Disable clustering since we want exact time slots
      lockType: "RESOURCES", // Lock the resources
      finishOrderBySlotEnd: true // Ensure order finishes within slot
    },
  };

  console.log('Fetching slots with data:', JSON.stringify(bookingData, null, 2));
```

```
   const response = await fetch(

"https://api.optimoroute.com/v1/booking_slots?key=0115400db6ed3e58f1c596bca2555ebf58fd
dkoP1Gc",
     {
       method: "POST",
       headers: {
         "Content-Type": "application/json",
       },
       body: JSON.stringify(bookingData),
     }
   );

   if (response.ok) {
     const responseData = await response.json();
     console.log("Full Response Data:", responseData);

     if (Array.isArray(responseData.slots)) {
       const allHaveCost = responseData.slots.every((slot: Slot) =>
slot.hasOwnProperty('cost'));

       if (allHaveCost) {
         console.log("All slots have the 'cost' property.");
       } else {
         console.warn("Some slots do not have the 'cost' property.");
       }
     } else {
       console.warn("No 'slots' array found in response data.");
     }

     return NextResponse.json(responseData, { status: 200 });
   } else {
     return NextResponse.json(
       {
         success: false,
         code: response.status,
         message: response.statusText,
       },
       { status: response.status }
     );
   }
 } catch (error) {
```

```
    console.error('Error:', error);
    return NextResponse.json(
      { success: false, message: "Internal server error" },
      { status: 500 }
    );
  }
}
```

//Make reservation.tsx

```
import { NextRequest, NextResponse } from "next/server";

// This function handles POST requests to make a reservation.
export async function POST(request: NextRequest) {
  try {
    const { reservationId, createOrderIfBookingFails } = await request.json();

    // Define the API key directly here
    const apiKey = "0115400db6ed3e58f1c596bca2555ebf58fddkoP1Gc";

    // Make the API call to OptimoRoute with the reservation data
    const response = await fetch(
      `https://api.optimoroute.com/v1/booking_reserve?key=${apiKey}`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          reservationId,
          createOrderIfBookingFails: false,
        }),
      }
    );

    if (response.ok) {
      const responseData = await response.json();
      console.log(responseData);
      return NextResponse.json(responseData);
    } else {
```

```
    const errorData = await response.json();
    console.log(errorData);
    return NextResponse.json(errorData, { status: response.status });
  }
 } catch (error) {
  return NextResponse.json(
    { success: false, message: "An error occurred.", error: error instanceof Error ?
error.message : String(error)  },
    { status: 500 }
  );
 }
}
```

// updateorder.tsx

```
import { NextRequest, NextResponse } from "next/server";

// This function handles the POST request to update an order.
export async function POST(req: NextRequest) {
 const apiKey = "0115400db6ed3e58f1c596bca2555ebf58fddkoP1Gc";
 const url = `https://api.optimoroute.com/v1/create_or_update_orders?key=${apiKey}`;

 try {
  // Get the request payload from the request body
  const payload = await req.json();

  console.log("Submitting payload to OptimoRoute API:", payload);

  // Make the POST request to the OptimoRoute API
  const response = await fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(payload),
  });

  if (response.ok) {
    // Parse the response and send it back to the client
    const data = await response.json();
```

```
    console.log("OptimoRoute API Response:", data);

    // Alert or log based on specific conditions
    if (data.success) {
      console.log("Order successfully created or updated.");
      return NextResponse.json({ success: true, data });
    } else {
      console.warn("Order creation/update failed:", data.message || "Unknown
error.");
      return NextResponse.json(
        {
          success: false,
          message: data.message || "Order creation/update failed.",
        },
        { status: 400 }
      );
    }
  } else {
    // Handle non-200 responses
    const errorData = await response.json();
    console.error("OptimoRoute API Error:", errorData);

    // Check for specific error codes and respond accordingly
    if (errorData.code === "ERR_INVALID_OR_EXPIRED_RESERVATION") {
      console.warn("Reservation is invalid or expired.");
      return NextResponse.json(
        {
          success: false,
          message: "The reservation ID is invalid or has expired. Please try
selecting a new slot.",
        },
        { status: 400 }
      );
    }

    return NextResponse.json(errorData, { status: response.status });
  }
} catch (error: any) {
  // Handle any errors that occurred during the request
  console.error("An unexpected error occurred:", error.message || error);
  return NextResponse.json(
```

```
      { success: false, message: "Internal Server Error: " + (error.message || "Unknown
error.") },
      { status: 500 }
    );
  }
}
```