## **Consultation Form & Email API Setup: Detailed Summary**

This document summarizes the typical steps and code structure involved in setting up a consultation form in a Next.js application that, upon submission, sends the collected data to you via email.

**Core Architecture:**

1. **Frontend Form:** A React component captures user input.
2. **API Route:** A Next.js API route receives the submitted form data.
3. **Email Service:** The API route uses an email service (like Nodemailer with an SMTP provider, or a transactional email service like SendGrid/Resend) to send the email.
4. **Validation:** Input validation is typically performed on both the client-side (for better UX) and server-side (for security and integrity).

### **1. Frontend Consultation Form**

This is the user-facing part where data is entered.

**Likely File(s) Changed/Created:**

* src/app/consultation/page.tsx (if the form is directly on the page)
* OR src/components/forms/ConsultationForm.tsx (if it's a reusable component, which is then imported into the consultation page)
* Schema for validation (optional but recommended): src/lib/validators/consultationSchema.ts (new file)
* UI components for toast notifications would be used (e.g., src/components/ui/use-toast.ts and src/components/ui/toaster.tsx if using Shadcn UI).

**Key Features & Libraries Used/Added:**

* **React (react):** For building the UI.
* **react-hook-form:** For managing form state, handling submissions, and integrating validation.
* **zod:** For schema-based validation.
* **@hookform/resolvers:** To connect Zod with react-hook-form.
* **UI Components (e.g., from shadcn/ui):** <Input />, <Textarea />, <Button />, <Label />.
* **fetch API:** To send the form data to your backend API route.

**Conceptual Code Snippet (ConsultationForm.tsx or within consultation/page.tsx):**

"use client"; // If using event handlers and state  
  
import React, { useState } from 'react';  
import { useForm, SubmitHandler } from 'react-hook-form';  
import { zodResolver } from '@hookform/resolvers/zod';  
import \* as z from 'zod';  
// Assuming you have Shadcn UI components or similar  
import { Button } from "@/components/ui/button";  
import { Input } from "@/components/ui/input";  
import { Textarea } from "@/components/ui/textarea";  
import { Label } from "@/components/ui/label";  
import { useToast } from "@/components/ui/use-toast"; // For user feedback  
  
// 1. Define the form schema with Zod  
const consultationFormSchema = z.object({  
 fullName: z.string().min(2, { message: "Full name must be at least 2 characters." }),  
 email: z.string().email({ message: "Please enter a valid email address." }),  
 companyName: z.string().optional(),  
 message: z.string().min(10, { message: "Message must be at least 10 characters." }),  
});  
  
type ConsultationFormValues = z.infer<typeof consultationFormSchema>;  
  
export function ConsultationForm() {  
 const { toast } = useToast();  
 const [isLoading, setIsLoading] = useState(false);  
  
 const {  
 register,  
 handleSubmit,  
 reset,  
 formState: { errors },  
 } = useForm<ConsultationFormValues>({  
 resolver: zodResolver(consultationFormSchema),  
 });  
  
 const onSubmit: SubmitHandler<ConsultationFormValues> = async (data) => {  
 setIsLoading(true);  
 try {  
 const response = await fetch('/api/consultation', { // Your API endpoint  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify(data),  
 });  
  
 if (!response.ok) {  
 const errorResult = await response.json();  
 throw new Error(errorResult.message || 'Something went wrong');  
 }  
  
 toast({  
 title: "Message Sent!",  
 description: "Thanks for reaching out. We'll be in touch soon.",  
 });  
 reset(); // Reset form fields  
 } catch (error: any) {  
 toast({  
 title: "Error Sending Message",  
 description: error.message || "Could not send your message. Please try again.",  
 variant: "destructive",  
 });  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 return (  
 <form onSubmit={handleSubmit(onSubmit)} className="space-y-6">  
 <div>  
 <Label htmlFor="fullName">Full Name</Label>  
 <Input id="fullName" {...register("fullName")} disabled={isLoading} />  
 {errors.fullName && <p className="text-sm text-destructive mt-1">{errors.fullName.message}</p>}  
 </div>  
 <div>  
 <Label htmlFor="email">Email Address</Label>  
 <Input id="email" type="email" {...register("email")} disabled={isLoading} />  
 {errors.email && <p className="text-sm text-destructive mt-1">{errors.email.message}</p>}  
 </div>  
 <div>  
 <Label htmlFor="companyName">Company Name (Optional)</Label>  
 <Input id="companyName" {...register("companyName")} disabled={isLoading} />  
 </div>  
 <div>  
 <Label htmlFor="message">Your Message/Inquiry</Label>  
 <Textarea id="message" {...register("message")} rows={5} disabled={isLoading} />  
 {errors.message && <p className="text-sm text-destructive mt-1">{errors.message.message}</p>}  
 </div>  
 <Button type="submit" disabled={isLoading}>  
 {isLoading ? "Sending..." : "Send Message"}  
 </Button>  
 </form>  
 );  
}

**Summary of Changes to Frontend:**

* A form component was created/integrated.
* A Zod schema was defined for client-side validation.
* react-hook-form was used for form state management and submission.
* An onSubmit handler was implemented to send a POST request to /api/consultation.
* User feedback mechanisms (loading state, toast notifications) were added.

### **2. Backend API Route**

This Next.js API route receives the form data and handles the email sending.

**Likely File Changed/Created:**

* src/app/api/consultation/route.ts (new file for App Router)

**Key Features & Libraries Used/Added:**

* **Next.js API Route Handler:** Standard POST function.
* **zod:** For server-side validation of the incoming data.
* **Email Library (e.g., nodemailer):** To construct and send the email.
* **Environment Variables:** To securely access email service credentials.

**Conceptual Code Snippet (src/app/api/consultation/route.ts):**

// src/app/api/consultation/route.ts  
import { NextRequest, NextResponse } from 'next/server';  
import \* as z from 'zod';  
import nodemailer from 'nodemailer'; // Or your chosen email sending library  
  
// Re-define the schema on the server for validation  
const consultationFormSchema = z.object({  
 fullName: z.string().min(2, { message: "Full name is required."}),  
 email: z.string().email({ message: "Invalid email address."}),  
 companyName: z.string().optional(),  
 message: z.string().min(10, { message: "Message must be at least 10 characters."}),  
});  
  
export async function POST(req: NextRequest) {  
 try {  
 const body = await req.json();  
  
 // 1. Validate the incoming data  
 const validationResult = consultationFormSchema.safeParse(body);  
 if (!validationResult.success) {  
 // Log detailed validation errors for debugging on the server  
 console.error("Form validation failed:", validationResult.error.flatten());  
 return NextResponse.json(  
 { message: "Invalid form data provided.", errors: validationResult.error.flatten().fieldErrors },  
 { status: 400 }  
 );  
 }  
  
 const { fullName, email, companyName, message } = validationResult.data;  
  
 // 2. Configure your email transporter  
 // These MUST come from environment variables for security  
 if (!process.env.SMTP\_HOST || !process.env.SMTP\_PORT || !process.env.SMTP\_USER || !process.env.SMTP\_PASSWORD || !process.env.YOUR\_RECEIVING\_EMAIL\_ADDRESS) {  
 console.error("SMTP environment variables are not set.");  
 return NextResponse.json({ message: "Email server not configured." }, { status: 500 });  
 }  
   
 const transporter = nodemailer.createTransport({  
 host: process.env.SMTP\_HOST,  
 port: Number(process.env.SMTP\_PORT),  
 secure: process.env.SMTP\_SECURE === 'true', // true for 465, false for other ports like 587  
 auth: {  
 user: process.env.SMTP\_USER,  
 pass: process.env.SMTP\_PASSWORD,  
 },  
 // Optional: Add timeout and debugging for nodemailer  
 // connectionTimeout: 5000, // 5 seconds  
 // greetingTimeout: 5000, // 5 seconds  
 // socketTimeout: 5000, // 5 seconds  
 // debug: process.env.NODE\_ENV === 'development', // Enable debug output in development  
 // logger: process.env.NODE\_ENV === 'development', // Log to console in development  
 });  
  
 // 3. Construct the email to be sent to you  
 const mailOptionsToYou = {  
 from: `"Consultation Form" <${process.env.EMAIL\_FROM\_ADDRESS || process.env.SMTP\_USER}>`,  
 to: process.env.YOUR\_RECEIVING\_EMAIL\_ADDRESS,   
 subject: `New Consultation Request from: ${fullName}`,  
 html: `  
 <h2>New Consultation Request</h2>  
 <p><strong>Name:</strong> ${fullName}</p>  
 <p><strong>Email:</strong> ${email}</p>  
 <p><strong>Company:</strong> ${companyName || 'N/A'}</p>  
 <hr>  
 <p><strong>Message:</strong></p>  
 <p>${message.replace(/\n/g, '<br>')}</p>  
 `,  
 };  
   
 // Optional: Construct a confirmation email to the user  
 const mailOptionsToUser = {  
 from: `"Fae Intelligence" <${process.env.EMAIL\_FROM\_ADDRESS || process.env.SMTP\_USER}>`,  
 to: email, // Send to the user's email  
 subject: "Thank You for Your Consultation Request with Fae Intelligence!",  
 html: `  
 <p>Dear ${fullName},</p>  
 <p>Thank you for reaching out to Fae Intelligence. We have received your consultation request and will review your message shortly.</p>  
 <p>We aim to respond within 1-2 business days.</p>  
 <p>Here's a copy of the message you sent:</p>  
 <blockquote>  
 <p><strong>Name:</strong> ${fullName}</p>  
 <p><strong>Email:</strong> ${email}</p>  
 <p><strong>Company:</strong> ${companyName || 'N/A'}</p>  
 <p><strong>Message:</strong> ${message.replace(/\n/g, '<br>')}</p>  
 </blockquote>  
 <p>Best regards,<br/>The Fae Intelligence Team</p>  
 <p><a href="${process.env.NEXT\_PUBLIC\_SITE\_URL || 'https://faeintelligence.com'}">Visit our website</a></p>  
 `,  
 };  
  
 // 4. Send the emails  
 await transporter.sendMail(mailOptionsToYou);  
 console.log("Notification email sent to admin.");  
   
 // await transporter.sendMail(mailOptionsToUser); // Uncomment to send confirmation to user  
 // console.log("Confirmation email sent to user.");  
  
 return NextResponse.json({ message: "Message sent successfully!" }, { status: 200 });  
  
 } catch (error: any) {  
 console.error("Error in /api/consultation POST handler:", error);  
 // Avoid sending detailed internal errors to the client in production  
 const errorMessage = error.message || "An unexpected error occurred.";  
 return NextResponse.json(  
 { message: "Failed to send message.", error: process.env.NODE\_ENV === 'development' ? errorMessage : "Please try again later." },  
 { status: 500 }  
 );  
 }  
}

**Summary of Changes to Backend API Route:**

* Created a Next.js API route at /api/consultation/route.ts.
* Implemented a POST handler.
* Added server-side validation using Zod.
* Configured nodemailer (or a similar library) using environment variables for SMTP settings.
* Constructed email content with the form data to be sent to YOUR\_RECEIVING\_EMAIL\_ADDRESS.
* Optionally, constructed and sent a confirmation email to the user.
* Added error handling and logging.

### **3. Environment Variables**

Sensitive credentials and configurations were stored in environment variables.

**File Changed/Created:**

* .env.local (new file at the project root, **must be added to .gitignore**)

**Content Example:**

# .env.local  
  
# --- EMAIL SENDING CONFIGURATION ---  
# Replace with your actual email provider's SMTP details  
SMTP\_HOST=smtp.yourprovider.com  
SMTP\_PORT=587 # or 465 if using SSL  
SMTP\_SECURE=false # true for port 465 (SSL), false for port 587 (TLS)  
SMTP\_USER=your-email-address@yourprovider.com # The email account to send from  
SMTP\_PASSWORD=your-email-account-password-or-app-password # For the SMTP\_USER account  
  
# --- EMAIL ADDRESSES ---  
# The email address where you want to receive the consultation form submissions  
YOUR\_RECEIVING\_EMAIL\_ADDRESS=your-personal-or-business-email@example.com   
# Optional: A "From" address for emails sent by the system (can be same as SMTP\_USER)  
EMAIL\_FROM\_ADDRESS="Fae Intelligence Contact <noreply@yourdomain.com>"   
  
# --- SITE CONFIGURATION ---  
# Public URL of your site, used for links in emails, metadata, etc.  
NEXT\_PUBLIC\_SITE\_URL=http://localhost:9002 # For local development  
# NEXT\_PUBLIC\_SITE\_URL=https://faeintelligence.com # For production

**Summary of Changes to Environment Variables:**

* Defined placeholders for SMTP server details, login credentials, and the recipient email address. These must be configured with your actual email service provider's information.

### **4. Package Installation**

The following packages were essential for this setup:

npm install react-hook-form zod @hookform/resolvers nodemailer  
# or  
yarn add react-hook-form zod @hookform/resolvers nodemailer

And for TypeScript support with Nodemailer:

npm install -D @types/nodemailer  
# or  
yarn add -D @types/nodemailer

**Summary of Package Changes:**

* Added react-hook-form, zod, @hookform/resolvers for form handling and validation.
* Added nodemailer for sending emails.

**Overall Process:**

1. A user fills out the **frontend form** on the /consultation page.
2. Client-side validation (via Zod and react-hook-form) provides immediate feedback.
3. Upon valid submission, the form data is sent via a POST request to the /api/consultation **API route**.
4. The API route validates the data again on the server-side.
5. If valid, the API route uses nodemailer (configured with your SMTP credentials from **environment variables**) to format and send an email containing the form submission details to your specified YOUR\_RECEIVING\_EMAIL\_ADDRESS.
6. Optionally, a confirmation email is sent to the user.
7. The API route returns a success or error response to the frontend, which then displays a toast message to the user.

This setup ensures that when a user submits the consultation form, you are notified via email with their details. Remember to replace placeholder values in .env.local with your actual service credentials and email addresses.