Guía Completa: Implementación de EmailJS en React

📋 Índice

Contexto y Objetivo

🎯 ¿Qué queríamos lograr?

- **Problema:** Formulario de contacto que solo mostraba datos en consola
- **Objetivo:** Enviar emails reales cuando alguien complete el formulario
- **Resultado:** Emails llegan directamente a Gmail del desarrollador

📧 Flujo de trabajo deseado:

```
Usuario llena formulario → Datos se envían a EmailJS → EmailJS envía email → Desarrollador recibe en Gmail
```

🏗️ Arquitectura de la solución:

- **Frontend:** React con formulario controlado
- **Servicio:** EmailJS como intermediario
- **Destino:** Gmail del desarrollador
- **Ventajas:** Sin backend, configuración simple, emails reales

---

Herramientas y Tecnologías

🛠️ Stack Tecnológico

Frontend:

- **React 18+** - Framework de JavaScript
- **useState** - Hook para manejo de estado
- **useTranslation** - Internacionalización (i18n)
- **Vite** - Build tool y servidor de desarrollo

Servicio de Email:

- **EmailJS** - Servicio de envío de emails desde JavaScript
- **Plan:** Gratuito (200 emails/mes)
- **Integración:** Directa desde frontend

Proveedor de Email:

- **Gmail** - Para recibir los emails
- **Configuración:** OAuth2 con EmailJS

📦 Dependencias instaladas:

```
npm install @emailjs/browser
```

🔧 Herramientas de desarrollo:

- **Chrome DevTools** - Para debugging
- **Console.log** - Para verificar flujo de datos
- **React DevTools** - Para inspeccionar componentes

Configuración de EmailJS

📋 Paso 1: Crear cuenta en EmailJS

Proceso:

1. **Ir a:** emailjs.com
2. **Registrarse** con email: `evablancomart@gmail.com`
3. **Confirmar cuenta** via email
4. **Acceder al dashboard**

¿Por qué EmailJS?

- ☑ **No requiere backend**
- ☑ **Configuración simple**
- ☑ **Integración directa con React**
- ☑ **Plan gratuito generoso**
- ☑ **Soporte para múltiples proveedores**

🔗 Paso 2: Configurar proveedor de email (Gmail)

Proceso:

1. **Dashboard → Email Services**
2. **"Add New Service" → Gmail**

3. **Conectar cuenta Gmail**
4. **Autorizar permisos de EmailJS**

Configuración obtenida:

```
Service ID: 'service_zr30ywk'
```

Permisos necesarios:

- **"Send email on your behalf"** - Para enviar emails
- **Acceso a Gmail API** - Para integración

📝 Paso 3: Crear template de email

Proceso:

1. **Dashboard → Email Templates**
2. **"Create New Template"**
3. **Configurar estructura del email**

Template configurado:

```
Subject: Nuevo mensaje desde portfolio - {{from_name}}

Content:
Nuevo mensaje recibido desde tu portfolio:

NOMBRE: {{from_name}}
EMAIL: {{from_email}}
MENSAJE: {{message}}

---
Enviado desde tu portfolio web
```

Variables del template:

- `{{from_name}}` - Nombre del remitente
- `{{from_email}}` - Email del remitente
- `{{message}}` - Mensaje del formulario

Configuración obtenida:

```
Template ID: 'template_xxxxxxxxx' // Se genera automáticamente
```

🔑 Paso 4: Obtener credenciales

Credenciales necesarias:

1. **Service ID:** `service_zr30ywk`
2. **Template ID:** `template_xxxxxxxxx`
3. **Public Key:** `75AKMtx-1o-R9J9SX`

Ubicación de credenciales:

- **Service ID:** Email Services → Gmail
- **Template ID:** Email Templates → Template específico
- **Public Key:** Account → API Keys

Implementación en React

📁 Estructura del proyecto:

```
src/
├── components/
│   ├── Contact.jsx      # Componente principal
│   └── Contact.css      # Estilos
├── locales/             # Archivos de traducción
└── i18n/               # Configuración i18n
```

🔧 Paso 1: Instalar EmailJS

Comando:

```
npm install @emailjs/browser
```

¿Por qué esta librería?

- ☑ **Oficial de EmailJS**
- ☑ **Optimizada para navegadores**
- ☑ **Soporte completo para React**
- ☑ **Manejo de errores robusto**

📝 Paso 2: Importar EmailJS

Código:

```
import emailjs from '@emailjs/browser';
```

Ubicación: Al inicio del archivo `Contact.jsx`

🎯 Paso 3: Añadir estados adicionales

Estados nuevos:

```
const [isSubmitting, setIsSubmitting] = useState(false); // Estado de carga
const [submitStatus, setSubmitStatus] = useState('idle'); // Estado de resultado
```

¿Por qué estos estados?

- `isSubmitting`: Para mostrar loading y deshabilitar formulario
- `submitStatus`: Para mostrar mensajes de éxito/error

🔄 Paso 4: Modificar función handleSubmit

Código completo:

```
const handleSubmit = (e) => {
    e.preventDefault();

    setIsSubmitting(true);
    setSubmitStatus('idle');

    // Configuración de EmailJS
    const templateParams = {
        from_name: formDataName,
        from_email: email,
        message: mensaje,
        title: 'Nuevo mensaje desde portfolio'
    };

    // Enviar email usando EmailJS
    emailjs.send(
        'service_zr30ywk',     // Service ID
        'template_xxxxxxxxx',   // Template ID
        templateParams,         // Datos del formulario
        '75AKMtx-1o-R9J9SX'    // Public Key
    )
    .then((response) => {
        console.log('Email enviado exitosamente:', response);
        setSubmitStatus('success');
        // Limpiar formulario
        setNombre('');
        setEmail('');
        setMensaje('');
    })
    .catch((error) => {
        console.error('Error al enviar email:', error);
```

```
            setSubmitStatus('error');
        })
        .finally(() => {
            setIsSubmitting(false);
        });
    };
```

Explicación línea por línea:

Prevención de envío automático:

```
e.preventDefault();
```

- **¿Por qué?** Evita que el formulario recargue la página
- **Sin esto:** La página se recarga y se pierden los datos

Estados de carga:

```
setIsSubmitting(true);
setSubmitStatus('idle');
```

- `isSubmitting(true)`**:** Activa el estado de carga
- `setSubmitStatus('idle')`**:** Resetea el estado de resultado

Parámetros del template:

```
const templateParams = {
    from_name: formDataName,
    from_email: email,
    message: mensaje,
    title: 'Nuevo mensaje desde portfolio'
};
```

- **Coinciden con variables del template:** `{{from_name}}`, `{{from_email}}`, `{{message}}`
- `title`**:** Para el asunto del email

Envío con EmailJS:

```
emailjs.send(
    'service_zr30ywk',      // Service ID
    'template_xxxxxxxxx',   // Template ID
    templateParams,         // Datos
```

```
    '75AKMtx-1o-R9J9SX'    // Public Key
)
```

- **Método:** `emailjs.send()`
- **Parámetros:** Service ID, Template ID, datos, Public Key

Manejo de respuesta:

```
.then((response) => {
    console.log('Email enviado exitosamente:', response);
    setSubmitStatus('success');
    // Limpiar formulario
    setNombre('');
    setEmail('');
    setMensaje('');
})
```

- `.then()`**:** Se ejecuta si el envío es exitoso
- `response`**:** Contiene información del envío
- **Limpieza:** Vuelve los campos a vacío

Manejo de errores:

```
.catch((error) => {
    console.error('Error al enviar email:', error);
    setSubmitStatus('error');
})
```

- `.catch()`**:** Se ejecuta si hay error
- `error`**:** Contiene detalles del error
- **Estado:** Marca como error para mostrar mensaje

Finalización:

```
.finally(() => {
    setIsSubmitting(false);
});
```

- `.finally()`**:** Se ejecuta siempre (éxito o error)
- **Propósito:** Desactiva el estado de carga

🎨 Paso 5: Mejorar UX del formulario

Campos deshabilitados durante envío:

```
<input
    // ... otras props
    disabled={isSubmitting}
/>
```

Botón con estado dinámico:

```
<button
    type="submit"
    className="btn-primary"
    disabled={isSubmitting}
>
    {isSubmitting ? 'Enviando...' : t('contact.form.submit')}
</button>
```

Mensajes de feedback:

```
{submitStatus === 'success' && (
    <div className="success-message">
        ☑ ¡Mensaje enviado con éxito! Te responderé pronto.
    </div>
)}
{submitStatus === 'error' && (
    <div className="error-message">
        ❌ Error al enviar el mensaje. Inténtalo de nuevo.
    </div>
)}
```

Solución de Problemas

🚨 Error 1: "The Public Key is invalid"

Síntomas:

```
Error 400: The Public Key is invalid. To find this ID, visit
https://dashboard.emailjs.com/admin/account
```

Causa:

- Public Key incorrecta o mal copiada
- Confusión con otras credenciales

Solución:

1. **Ir a:** Account → API Keys
2. **Copiar Public Key completa**
3. **Verificar formato:** `75AKMtx-1o-R9J9SX`

Prevención:

- Copiar credenciales una por una
- Verificar que no haya espacios extra
- Confirmar que sea la credencial correcta

🚨 Error 2: "The template ID not found"

Síntomas:

```
Error 400: The template ID not found. To find this ID, visit
https://dashboard.emailjs.com/admin/templates
```

Causa:

- Template ID incorrecto
- Template eliminado o no guardado
- Template de otra cuenta

Solución:

1. **Verificar templates existentes**
2. **Crear template nuevo si es necesario**
3. **Copiar Template ID correcto**

Prevención:

- Verificar que el template esté guardado
- Usar Template ID de la cuenta correcta
- Confirmar que el template esté activo

🚨 Error 3: "Gmail_API: Request had insufficient authentication scopes"

Síntomas:

```
Error 412: Gmail_API: Request had insufficient authentication scopes
```

Causa:

- Permisos insuficientes en Gmail
- Configuración OAuth incompleta

Solución:

1. **Desconectar y reconectar Gmail**
2. **Asegurar permisos completos**
3. **Autorizar "Send email on your behalf"**

Prevención:

- Dar todos los permisos solicitados
- Usar cuenta Gmail personal (no corporativa)
- Verificar configuración OAuth

🔍 Debugging efectivo:

Console.log estratégicos:

```
console.log('Datos del formulario:', templateParams);
console.log('Credenciales:', {
    serviceId: 'service_zr30ywk',
    templateId: 'template_xxxxxxxxx',
    publicKey: '75AKMtx-1o-R9J9SX'
});
```

Verificación de credenciales:

1. **Service ID:** Debe empezar con `service_`
2. **Template ID:** Debe empezar con `template_`
3. **Public Key:** Formato alfanumérico

Verificación y Testing

☑️ Paso 1: Verificar configuración

Checklist:

- ☐ EmailJS cuenta creada y verificada
- ☐ Gmail conectado y autorizado
- ☐ Template creado y guardado
- ☐ Credenciales copiadas correctamente
- ☐ Código actualizado con credenciales

🧪 Paso 2: Testing del formulario

Datos de prueba:

```
Nombre: "Eva Blanco"
Email: "evablancomart@gmail.com"
Mensaje: "Este es un mensaje de prueba desde mi portfolio"
```

Verificación visual:

- ☐ Campos se llenan correctamente
- ☐ Botón cambia a "Enviando..."
- ☐ Campos se deshabilitan durante envío
- ☐ Aparece mensaje de éxito/error
- ☐ Formulario se limpia después del éxito

📧 Paso 3: Verificar recepción de email

Ubicaciones a revisar:

1. **Bandeja principal** de Gmail
2. **Carpeta de spam** (más común)
3. **Filtros de Gmail**
4. **Búsqueda por remitente:** `noreply@emailjs.com`

Contenido esperado:

```
Asunto: Nuevo mensaje desde portfolio - Eva Blanco

NOMBRE: Eva Blanco
EMAIL: evablancomart@gmail.com
MENSAJE: Este es un mensaje de prueba desde mi portfolio

---
Enviado desde tu portfolio web
```

🔍 Paso 4: Verificación en consola

Mensajes de éxito:

```
Email enviado exitosamente: EmailJSResponseStatus {status: 200, text: 'OK'}
```

Mensajes de error:

```
Error al enviar email: EmailJSResponseStatus {status: 400, text: 'Error
específico'}
```

Mejoras y Optimizaciones

Estilos para mensajes de estado

CSS recomendado:

```
.success-message {
    background-color: #d4edda;
    color: #155724;
    padding: 12px;
    border-radius: 8px;
    margin: 16px 0;
    border: 1px solid #c3e6cb;
}

.error-message {
    background-color: #f8d7da;
    color: #721c24;
    padding: 12px;
    border-radius: 8px;
    margin: 16px 0;
    border: 1px solid #f5c6cb;
}
```

Variables de entorno

Crear archivo `.env`:

```
VITE_EMAILJS_SERVICE_ID=service_zr30ywk
VITE_EMAILJS_TEMPLATE_ID=template_xxxxxxxxx
VITE_EMAILJS_PUBLIC_KEY=75AKMtx-1o-R9J9SX
```

Usar en código:

```
emailjs.send(
    import.meta.env.VITE_EMAILJS_SERVICE_ID,
    import.meta.env.VITE_EMAILJS_TEMPLATE_ID,
    templateParams,
    import.meta.env.VITE_EMAILJS_PUBLIC_KEY
)
```

Ventajas:

- ☑ **Seguridad:** Credenciales no en código

- ☑ **Flexibilidad:** Fácil cambio entre entornos
- ☑ **Buenas prácticas:** Estándar de la industria

🧮 Mejoras de UX

Loading spinner:

```
{isSubmitting && (
    <div className="loading-spinner">
        <div className="spinner"></div>
        <p>Enviando mensaje...</p>
    </div>
)}
```

Validación en tiempo real:

```
const [errors, setErrors] = useState({});

const validateField = (name, value) => {
    const newErrors = { ...errors };

    if (name === 'email' && !value.includes('@')) {
        newErrors.email = 'Email inválido';
    } else {
        delete newErrors.email;
    }

    setErrors(newErrors);
};
```

Auto-hide mensajes:

```
useEffect(() => {
    if (submitStatus === 'success' || submitStatus === 'error') {
        const timer = setTimeout(() => {
            setSubmitStatus('idle');
        }, 5000);

        return () => clearTimeout(timer);
    }
}, [submitStatus]);
```

🚀 Optimizaciones de rendimiento

Debounce para validación:

```
import { useCallback } from 'react';
import { debounce } from 'lodash';

const debouncedValidate = useCallback(
    debounce((name, value) => {
        validateField(name, value);
    }, 300),
    []
);
```

Memoización de funciones:

```
const handleSubmit = useCallback((e) => {
    // ... lógica del submit
}, [formDataName, email, mensaje]);
```

Resumen y Lecciones Aprendidas

☑ Lo que logramos:

Funcionalidad completa:

- ☑ Formulario de contacto funcional
- ☑ Envío real de emails
- ☑ Recepción en Gmail
- ☑ Feedback visual al usuario
- ☑ Manejo de errores robusto

Experiencia de usuario:

- ☑ Estados de carga claros
- ☑ Mensajes de éxito/error
- ☑ Formulario se limpia automáticamente
- ☑ Campos deshabilitados durante envío

🎓 Lecciones aprendidas:

1. Importancia de las credenciales correctas:

- **Problema:** Public Key incorrecta causó error 400
- **Solución:** Verificar cada credencial individualmente
- **Lección:** Siempre verificar credenciales antes de implementar

2. Debugging sistemático:

- **Problema:** Múltiples errores en secuencia
- **Solución:** Resolver un error a la vez
- **Lección:** Enfocarse en un problema específico

3. Verificación de configuración:

- **Problema:** Template no existía en la cuenta
- **Solución:** Crear template nuevo
- **Lección:** Verificar que todos los recursos existan

4. Testing completo:

- **Problema:** Código funcionaba pero email no llegaba
- **Solución:** Verificar spam y configuración
- **Lección:** Probar todo el flujo, no solo el código

🚀 Aplicación en otros proyectos:

Patrón reutilizable:

```
// 1. Configurar EmailJS
// 2. Obtener credenciales
// 3. Crear template
// 4. Implementar en React
// 5. Añadir estados de UX
// 6. Manejar errores
// 7. Verificar funcionamiento
```

Componentes reutilizables:

- **Formulario de contacto** con EmailJS
- **Estados de carga** y feedback
- **Manejo de errores** estándar
- **Validación** de formularios

Configuración estándar:

- **Variables de entorno** para credenciales
- **Templates** reutilizables
- **Estilos** consistentes
- **Testing** automatizado

📚 Recursos adicionales:

Documentación oficial:

- EmailJS Documentation

- React EmailJS Integration
- Troubleshooting Guide

Herramientas útiles:

- EmailJS Dashboard
- Gmail API Documentation
- React DevTools

🎯 Conclusión

Esta implementación demuestra cómo integrar servicios externos en aplicaciones React de manera efectiva. El proceso incluye:

1. **Análisis del problema** y selección de herramientas
2. **Configuración sistemática** del servicio externo
3. **Implementación gradual** con testing continuo
4. **Solución de problemas** metodológica
5. **Mejoras de UX** y optimizaciones
6. **Documentación completa** para futuras referencias

El resultado es un formulario de contacto completamente funcional que envía emails reales, proporcionando una experiencia profesional tanto para el usuario como para el desarrollador.

Esta guía sirve como referencia completa para implementar EmailJS en cualquier proyecto React, siguiendo las mejores prácticas y aprendiendo de los errores comunes.