**Email Service API - Complete Documentation**

**📖 Overview**

A reusable, multi-tenant email service that provides RESTful APIs for sending various types of emails. Built with Node.js, Express, and Nodemailer, this service allows you to integrate email functionality into any project without rebuilding email infrastructure.

**🚀 Features**

✅ **Multi-tenant Architecture** - Each user provides their own SMTP credentials

✅ **Pre-built Email Templates** - Contact forms, verification, password reset, welcome emails

✅ **Custom Email Support** - Send any type of email with custom content

✅ **Template Engine** - EJS templates with dynamic data

✅ **Bulk Email Support** - Send to multiple recipients

✅ **File Attachments** - Support for base64 encoded attachments

✅ **SMTP Testing** - Validate SMTP configuration before sending

✅ **API Key Authentication** - Secure your email service

✅ **CORS Enabled** - Use from any frontend application

✅ **Deployment Ready** - Optimized for Vercel, Heroku, etc.

**🏗️ Architecture**

Email Service Architecture:

Client App → Email Service API → SMTP Server → Recipient

↓ ↓ ↓

Provides Processes Delivers

SMTP Config & Sends Email

**📋 Use Cases**

**1. Authentication Systems**

* Email verification for new users
* Password reset functionality
* Welcome emails

**2. Contact Forms**

* Portfolio websites
* Business websites
* Customer support systems

**3. Notifications**

* Order confirmations
* Payment receipts
* System alerts
* Newsletter campaigns

**4. Marketing**

* Welcome sequences
* Product updates
* Promotional campaigns

**🔧 Installation & Setup**

**Prerequisites**

* Node.js 18+
* SMTP account (Gmail, SendGrid, Mailgun, etc.)

**Local Development**

*# Clone and setup*

git clone <your-repo>

cd email-service

npm install

*# Environment setup*

cp .env.example .env

*# Edit .env with your settings*

*# Start development server*

npm run dev

**Environment Variables**

# Server

NODE\_ENV=development

PORT=3000

CORS\_ORIGIN=\*

# Security

API\_KEY=your-secret-api-key-12345

**🚀 Deployment**

**Vercel Deployment**

*# Install Vercel CLI*

npm install -g vercel

*# Deploy*

vercel

*# Set environment variables in Vercel dashboard*

API\_KEY=your-production-api-key

CORS\_ORIGIN=\*

**Environment Variables in Production**

Set these in your hosting platform:

* API\_KEY: Your secret API key for authentication
* CORS\_ORIGIN: Allowed origins (use \* for development)

**📚 API Reference**

**Base URL**

* **Production**: https://your-app.vercel.app
* **Development**: http://localhost:3000

**Authentication**

All endpoints (except health check) require API key in header:

x-api-key: your-secret-api-key-12345

**🔍 Endpoints**

**1. Health Check**

**GET** /health

Check if the service is running.

**Response:**

json

{

"success": true,

"message": "Email service is healthy",

"timestamp": "2024-01-15T10:30:00.000Z",

"environment": "production"

}

**2. Test SMTP Configuration**

**POST** /api/email/test-smtp

Validate SMTP credentials before sending emails.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

}

}

**Response:**

json

{

"success": true,

"message": "SMTP configuration is valid and working!",

"user": "your-email@gmail.com"

}

**3. Send Generic Email**

**POST** /api/email/send

Send any type of email with full customization.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

},

"to": "recipient@example.com",

"subject": "Your Email Subject",

"text": "Plain text content",

"html": "<h1>HTML content</h1>",

"from": "custom@yourapp.com",

"cc": "cc@example.com",

"bcc": "bcc@example.com",

"replyTo": "reply@example.com",

"template": "welcome",

"templateData": {

"name": "John Doe",

"verificationUrl": "https://yourapp.com/verify"

},

"attachments": [

{

"filename": "document.pdf",

"content": "base64encodedstring",

"encoding": "base64"

}

]

}

**Parameters:**

* to (required): Single email or array of emails
* subject (required): Email subject
* text or html or template (required): Email content
* templateData: Data for template rendering
* attachments: Array of file attachments

**4. Send Contact Form Email**

**POST** /api/email/contact

Specifically designed for contact forms.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

},

"contactEmail": "admin@yourapp.com",

"name": "John Doe",

"email": "john@example.com",

"message": "Hello, I'm interested in your services...",

"subject": "Contact Form Submission"

}

**Response:**

json

{

"success": true,

"message": "Contact email sent successfully",

"data": {

"messageId": "<d123456789@email-service>"

}

}

**5. Send Verification Email**

**POST** /api/email/verification

For email verification in authentication systems.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

},

"to": "user@example.com",

"name": "Alice Smith",

"verificationUrl": "https://yourapp.com/verify?token=abc123"

}

**6. Send Password Reset Email**

**POST** /api/email/password-reset

For password reset functionality.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

},

"to": "user@example.com",

"name": "Bob Wilson",

"resetUrl": "https://yourapp.com/reset-password?token=xyz789"

}

**7. Send Welcome Email**

**POST** /api/email/welcome

Welcome new users to your platform.

**Request:**

json

{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

},

"to": "newuser@example.com",

"name": "Sarah Johnson"

}

**🎨 Available Templates**

**1. contact**

* **Use**: Contact form submissions
* **Variables**: name, email, message, subject

**2. verification**

* **Use**: Email verification
* **Variables**: name, verificationUrl, expiryTime

**3. password-reset**

* **Use**: Password reset requests
* **Variables**: name, resetUrl, expiryTime

**4. welcome**

* **Use**: Welcome new users
* **Variables**: name

**5. notification**

* **Use**: System notifications
* **Variables**: subject, message, priority

**🔐 SMTP Configuration**

**Supported Providers**

**Gmail**

json

{

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

}

**Outlook/Office365**

json

{

"host": "smtp-mail.outlook.com",

"port": 587,

"secure": false,

"user": "your-email@outlook.com",

"pass": "your-password"

}

**SendGrid**

json

{

"host": "smtp.sendgrid.net",

"port": 587,

"secure": false,

"user": "apikey",

"pass": "your-sendgrid-api-key"

}

**Mailgun**

json

{

"host": "smtp.mailgun.org",

"port": 587,

"secure": false,

"user": "postmaster@your-domain.com",

"pass": "your-mailgun-password"

}

**Custom SMTP**

json

{

"host": "your-smtp-server.com",

"port": 587,

"secure": false,

"user": "your-username",

"pass": "your-password"

}

**💻 Client Integration Examples**

**Frontend (React)**

javascript

import { useState } from 'react';

const EmailService = () => {

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

const sendContactEmail = async (formData) => {

setLoading(true);

try {

const response = await fetch('https://your-app.vercel.app/api/email/contact', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'x-api-key': 'your-api-key'

},

body: JSON.stringify({

smtpConfig: {

host: 'smtp.gmail.com',

port: 587,

secure: false,

user: 'your-email@gmail.com',

pass: 'your-app-password'

},

contactEmail: 'admin@yourapp.com',

...formData

})

});

const result = await response.json();

return result;

} finally {

setLoading(false);

}

};

return { sendContactEmail, loading };

};

export default EmailService;

**Backend (Node.js)**

javascript

class EmailClient {

constructor(baseURL, apiKey) {

this.baseURL = baseURL;

this.apiKey = apiKey;

}

async sendVerificationEmail(userEmail, userName, verificationToken) {

const response = await fetch(`${this.baseURL}/api/email/verification`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'x-api-key': this.apiKey

},

body: JSON.stringify({

smtpConfig: {

host: process.env.SMTP\_HOST,

port: process.env.SMTP\_PORT,

secure: false,

user: process.env.SMTP\_USER,

pass: process.env.SMTP\_PASS

},

to: userEmail,

name: userName,

verificationUrl: `https://yourapp.com/verify?token=${verificationToken}`

})

});

return await response.json();

}

async sendPasswordResetEmail(userEmail, userName, resetToken) {

const response = await fetch(`${this.baseURL}/api/email/password-reset`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'x-api-key': this.apiKey

},

body: JSON.stringify({

smtpConfig: {

host: process.env.SMTP\_HOST,

port: process.env.SMTP\_PORT,

secure: false,

user: process.env.SMTP\_USER,

pass: process.env.SMTP\_PASS

},

to: userEmail,

name: userName,

resetUrl: `https://yourapp.com/reset-password?token=${resetToken}`

})

});

return await response.json();

}

}

module.exports = EmailClient;

**Portfolio Contact Form**

javascript

*// portfolio-website contact form*

const handleContactSubmit = async (e) => {

e.preventDefault();

const response = await fetch('https://your-email-service.vercel.app/api/email/contact', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'x-api-key': 'your-portfolio-api-key'

},

body: JSON.stringify({

smtpConfig: {

host: 'smtp.gmail.com',

port: 587,

secure: false,

user: 'portfolio@yourdomain.com',

pass: 'your-app-password'

},

contactEmail: 'your-personal@gmail.com',

name: formData.name,

email: formData.email,

message: formData.message,

subject: 'Portfolio Contact Form'

})

});

const result = await response.json();

if (result.success) {

alert('Message sent successfully!');

} else {

alert('Failed to send message: ' + result.error);

}

};

**🛠️ Error Handling**

**Common Error Responses**

json

{

"success": false,

"error": "Error description"

}

**Error Types:**

* 400 - Bad Request (missing parameters, invalid email)
* 401 - Unauthorized (missing/invalid API key)
* 500 - Internal Server Error (SMTP issues, server errors)

**SMTP Common Errors:**

* Invalid login - Check SMTP credentials
* Connection timeout - Check SMTP host/port
* Message rejected - Check recipient email

**🔒 Security**

**API Key Protection**

* Always use environment variables for API keys
* Rotate API keys regularly
* Use different API keys for different environments

**SMTP Security**

* Use app passwords instead of main account passwords
* Enable 2-factor authentication on email accounts
* Regularly monitor email sending patterns

**📊 Monitoring & Logging**

The service provides built-in logging:

* SMTP connection attempts
* Email sending successes/failures
* API request logging

**Recommended Monitoring:**

* Set up health check monitoring
* Monitor error rates
* Track email delivery rates
* Set up alerts for service downtime

**🚀 Performance Tips**

**1. Connection Pooling**

The service uses connection pooling for better performance with multiple emails.

**2. Template Caching**

Templates are cached in memory for faster rendering.

**3. Bulk Sending**

Use array of recipients for bulk emails instead of individual calls.

**4. Async Processing**

All email sending is non-blocking and asynchronous.

**🔄 Versioning**

**Current Version**: v1.0.0

All endpoints are versioned. Future updates will maintain backward compatibility.

**🤝 Contributing**

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests
5. Submit a pull request

**📄 License**

MIT License - feel free to use in personal and commercial projects.

**🆘 Support**

**Common Issues:**

1. **SMTP Connection Failed**
   * Verify SMTP credentials
   * Check if app passwords are required
   * Test SMTP configuration first
2. **Emails Going to Spam**
   * Configure proper SPF/DKIM records
   * Use consistent from addresses
   * Avoid spammy content
3. **API Timeouts**
   * Check SMTP server response times
   * Optimize template rendering
   * Use connection pooling

**Getting Help:**

* Check the API responses for specific error messages
* Test with the /test-smtp endpoint first
* Verify all required parameters are included

**🎯 Quick Start Guide**

**1. Deploy the Service**

bash

vercel deploy

**2. Set Environment Variables**

In Vercel dashboard:

text

API\_KEY=your-secure-api-key

CORS\_ORIGIN=\*

**3. Test SMTP Configuration**

bash

curl -X POST https://your-app.vercel.app/api/email/test-smtp \

-H "Content-Type: application/json" \

-H "x-api-key: your-api-key" \

-d '{

"smtpConfig": {

"host": "smtp.gmail.com",

"port": 587,

"secure": false,

"user": "your-email@gmail.com",

"pass": "your-app-password"

}

}'

**4. Start Sending Emails!**

This documentation provides everything needed to integrate the email service into any project. The service is production-ready and can handle all common email requirements.