**validateEmailN\_R API Documentation**

**Overview**

This API provides functionalities to handle email validation and OTP (One-Time Password) generation and validation for registered users.

* **Lambda Function Name:** validateEmailN\_R
* **API Name:** Module1\_Refactored
* **Resource Name:** validateEmailN\_R
* **API URL:** <https://aymk6mhnl3.execute-api.us-east-1.amazonaws.com/v1/validateEmailN_R>

**Endpoints**

**1. Validate Email and OTP**

* **Description:** Validates the email format and performs OTP generation or validation.
* **Endpoint:** POST /validateEmailN\_R
* **Request Body:**

json

Copy code

{

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

"action": "generate", // or "validate"

"otp": "123456" // Required for "validate" action

}

* **Response Codes:**
  + 200 OK: Successful operation
  + 400 Bad Request: Invalid email format, missing OTP, or invalid OTP
  + 404 Not Found: User not registered
  + 500 Internal Server Error: Server encountered an error
* **Sample Request:**

bash

Copy code

curl -X POST https://aymk6mhnl3.execute-api.us-east-1.amazonaws.com/v1/validateEmailN\_R \

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

-d '{"email": "user@example.com", "action": "generate"}'

* **Sample Response (200 OK):**

json

Copy code

{

"message": "OTP sent to email"

}

* **Sample Response (400 Bad Request):**

json

Copy code

{

"message": "Invalid OTP"

}

**2. Error Responses**

* **Response (404 Not Found):**

json

Copy code

{

"message": "User not registered"

}

* **Response (500 Internal Server Error):**

json

Copy code

{

"message": "Internal server error"

}

**Implementation Details**

* **Dependencies:**
  + nodemailer: Used for sending emails via SMTP.
  + googleapis: Used for OAuth2 authentication with Gmail.
  + mongoose: MongoDB ORM for managing User model.
* **Functionality:**
  + Validates email format using regex.
  + Generates a random OTP and sends it via email.
  + Stores OTP in-memory for validation within a 15-minute window.
  + Validates OTP entered by the user.

**Security**

* **Authentication:** Uses OAuth2 with Gmail credentials (clientId, clientSecret, refreshToken) for secure email sending.
* **Authorization:** No specific authorization mechanism described. Ensure proper access controls are implemented at AWS API Gateway level.

**Notes**

* Ensure environment variables (EMAIL\_USER, CLIENT\_ID, CLIENT\_SECRET, REFRESH\_TOKEN, MONGO\_URL) are properly configured for the Lambda function to work correctly.

CODE:  
const nodemailer = require('nodemailer');

const { google } = require('googleapis');

const mongoose = require('mongoose');

const User = mongoose.model('User2', {

email: String,

});

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

type: 'OAuth2',

user: process.env.EMAIL\_USER,

clientId: process.env.CLIENT\_ID,

clientSecret: process.env.CLIENT\_SECRET,

refreshToken: process.env.REFRESH\_TOKEN, //update refresh token from https://developers.google.com/oauthplayground/ if says internal server error

}

});

let otpCache = {}; // In-memory storage for OTPs (for demonstration purposes)

mongoose.connect(process.env.MONGO\_URL);

const headers = {

'Access-Control-Allow-Origin': 'https://admin.exambuilder.online', // Allow requests from specific origin

'Access-Control-Allow-Methods': 'OPTIONS, POST', // Allow OPTIONS and POST methods

'Access-Control-Allow-Headers': 'Content-Type' // Allow Content-Type header

};

exports.handler = async (event) => {

// Handle preflight (OPTIONS) request

if (event.httpMethod === 'OPTIONS') {

return {

statusCode: 204,

headers: headers

};

}

try {

const { email, action, otp } = JSON.parse(event.body);

const lowercaseEmail = email.toLowerCase();

// Email format validation using regex

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailRegex.test(lowercaseEmail)) {

return {

statusCode: 400,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'Invalid email format' }),

};

}

const user = await User.findOne({ email: lowercaseEmail });

if (!user) {

return {

statusCode: 404,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'User not registered' }),

};

}

if (action === 'generate') {

// Generate OTP and send via email

const generatedOtp = await generateOtp(lowercaseEmail);

return {

statusCode: 200,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'OTP sent to email' }),

};

} else if (action === 'validate') {

// Validate entered OTP

const isValid = validateOtp(lowercaseEmail, otp);

if (isValid) {

delete otpCache[lowercaseEmail]; // Empty the stored OTP after successful validation

return {

statusCode: 200,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'OTP verified successfully' }),

};

} else {

return {

statusCode: 400,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'Invalid OTP' }),

};

}

} else {

return {

statusCode: 400,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'Invalid action' }),

};

}

} catch (error) {

if (error instanceof SyntaxError) {

return {

statusCode: 400,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'Invalid JSON payload' }),

};

}

console.error('Error occurred:', error);

return {

statusCode: 500,

headers: { 'Access-Control-Allow-Origin': headers['Access-Control-Allow-Origin'] },

body: JSON.stringify({ message: 'Internal server error' }),

};

}

};

async function generateOtp(lowercaseEmail) {

// Generate OTP

const otp = Math.floor(100000 + Math.random() \* 900000).toString();

// Send OTP via email

try {

const mailOptions = {

from: process.env.EMAIL\_USER,

to: lowercaseEmail,

subject: 'Your One-Time Password (OTP)',

text: `Your OTP is ${otp}. It is valid for the next 15 minutes.`

};

const info = await transporter.sendMail(mailOptions);

console.log('Email sent:', info.response);

// Store OTP in cache (for validation)

otpCache[lowercaseEmail] = otp;

setTimeout(() => {

delete otpCache[lowercaseEmail]; // Remove OTP after expiry (e.g., 15 minutes)

}, 15 \* 60 \* 1000); // 15 minutes expiry time

return otp;

} catch (error) {

console.error('Failed to send email:', error);

throw new Error('Failed to send OTP via email');

}

}

function validateOtp(lowercaseEmail, enteredOtp) {

// Retrieve stored OTP from cache

const storedOtp = otpCache[lowercaseEmail];

// Validate entered OTP

return storedOtp && storedOtp === enteredOtp;

}