**validateNewEmailN\_R API Documentation**

**Lambda Function Details**

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

**Overview**

This API provides functionality to manage email validation and OTP (One-Time Password) verification using Google OAuth2 for sending emails.

**Endpoints**

**validateNewEmailN\_R**

* **Endpoint:** /v1/validateNewEmailN\_R
* **Method:** POST
* **Description:** Validates email actions such as generating OTP and verifying OTP.
* **Headers:**
  + Content-Type: application/json
* **Request Body:**

json

{

"email": "string",

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

"otp": "string" // Required for 'validate' action

}

* **Responses:**
  + **200 OK**

json

{

"message": "Success message"

}

* + **400 Bad Request**

json

{

"error": "Bad Request",

"message": "Error message"

}

* + **409 Conflict**

json

{

"error": "Conflict",

"message": "Email already registered"

}

* + **500 Internal Server Error**

json

{

"error": "Internal Server Error",

"message": "Something went wrong"

}

**Usage**

1. **Generate OTP**
   * **Description:** Generates and sends an OTP to the provided email address.
   * **Request:**

json

{

"email": "seniorporwal@gmail.com",

"action": "generate"

}

* + **Response (200 OK):**

json

{

"message": "OTP sent to email"

}

1. **Validate OTP**
   * **Description:** Validates the OTP provided for the email address.
   * **Request:**

json

{

"email": "seniorporwal@gmail.com",

"action": "validate",

"otp":"861341"

}

* + **Response (200 OK):**

json

{

"message": "OTP verified successfully"

}

**Notes**

* The API uses Google OAuth2 for authentication and authorization.
* MongoDB is used for storing user email data.
* OTPs are stored temporarily in memory (otpCache) and expire after 15 minutes.

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

const nodemailer = require('nodemailer');

const mongoose = require('mongoose');

const OAuth2 = google.auth.OAuth2;

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

email: String,

});

const oauth2Client = new OAuth2(

process.env.CLIENT\_ID,

process.env.CLIENT\_SECRET,

"https://developers.google.com/oauthplayground" // Redirect URL

);

oauth2Client.setCredentials({

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

});

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,

accessToken: oauth2Client.getAccessToken()

}

});

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

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

.then(() => console.log('MongoDB connected'))

.catch(err => {

console.error('MongoDB connection error:', err);

process.exit(1); // Exit process on MongoDB connection error

});

exports.handler = async (event) => {

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

return {

statusCode: 200,

headers: {

'Access-Control-Allow-Origin': 'https://admin.exambuilder.online',

'Access-Control-Allow-Methods': 'OPTIONS, POST',

'Access-Control-Allow-Headers': 'Content-Type',

},

};

}

try {

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

if (!email || !action) {

return createResponse(400, { error: 'Bad Request', message: 'Email or action missing' });

}

const lowerCaseEmail = email.toLowerCase();

// Email format validation

if (!validateEmailFormat(lowerCaseEmail)) {

return createResponse(400, { error: 'Bad Request', message: 'Invalid email format' });

}

if (action === 'generate') {

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

if (user) {

return createResponse(409, { error: 'Conflict', message: 'Email already registered' });

}

const generatedOtp = await generateOtp(lowerCaseEmail);

return createResponse(200, { message: 'OTP sent to email' });

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

if (!otp) {

return createResponse(400, { error: 'Bad Request', message: 'OTP missing' });

}

const isValid = validateOtp(lowerCaseEmail, otp);

if (isValid) {

// Clear OTP after successful validation

delete otpCache[lowerCaseEmail];

return createResponse(200, { message: 'OTP verified successfully' });

} else {

return createResponse(400, { error: 'Bad Request', message: 'Invalid OTP' });

}

} else {

return createResponse(400, { error: 'Bad Request', message: 'Invalid action' });

}

} catch (error) {

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

if (error instanceof SyntaxError) {

return createResponse(400, { error: 'Bad Request', message: 'Invalid JSON payload' });

}

return createResponse(500, { error: 'Internal Server Error', message: 'Something went wrong' });

}

};

const headers = {

'Access-Control-Allow-Origin': 'https://admin.exambuilder.online',

};

function createResponse(statusCode, body) {

return {

statusCode: statusCode,

headers: headers,

body: JSON.stringify(body),

};

}

// Email format validation function

function validateEmailFormat(email) {

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

return emailRegex.test(email);

}

async function generateOtp(lowerCaseEmail) {

try {

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

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);

otpCache[lowerCaseEmail] = otp;

setTimeout(() => {

delete otpCache[lowerCaseEmail];

}, 15 \* 60 \* 1000);

return otp;

} catch (error) {

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

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

}

}

function validateOtp(lowerCaseEmail, enteredOtp) {

const storedOtp = otpCache[lowerCaseEmail];

return storedOtp && storedOtp === enteredOtp;

}