**API’s LAMBDA FUN NAME -> forgetpasswordN (For Registration page)**

API’s URL -> <https://ejy88n4hr6.execute-api.us-east-1.amazonaws.com/email/verification>

Resource -> **NewRegisterUser ->** [**verification**](https://us-east-1.console.aws.amazon.com/apigateway/main/apis/ejy88n4hr6/resources?api=ejy88n4hr6&region=us-east-1)

Stage -> email

**1. Role of the API**

This API facilitates the generation and validation of One-Time Passwords (OTPs) for user authentication. It interacts with a MongoDB database to check if a user email is registered and utilizes an email service (Gmail in this case) to send OTPs to users. The API also includes logic to validate OTPs entered by users.

**2. Functioning**

The function processes POST requests with specified actions (**generate** or **validate**) along with user details. It first checks if the user email is registered in the database. If not, it returns an error response. If the email is registered, it performs the requested action: generating an OTP and sending it via email, or validating an entered OTP against the stored one.

**3. Request Body**

The API expects a JSON object with the following fields:

* **email**: The user's email address.
* **action**: The desired action, either **generate** or **validate**.
* **otp**: Required for the **validate** action; it is the OTP entered by the user.

Example of a request body for generating an OTP:

{

"body": "{\"email\": \"shivanisingh0fd31202@gmail.com\", \"action\": \"generate\"}"

}

Example of a request body for validating an OTP

{

"body": "{\"email\": \"shivanisingh0fd31202@gmail", \"action\": \"validate\", \"otp\": \"391159\"}"

}

### 4. Response

The API provides different responses based on the action and outcome:

* **200 OK**: For successful OTP generation or validation.
* **400 Bad Request**: If the action is invalid or if the entered OTP is incorrect.
* **404 Not Found**: If the user email is not registered.
* **500 Internal Server Error**: In case of errors during processing.

Sample response for successful OTP generation:

{

"statusCode": 200,

"headers": {

"Access-Control-Allow-Origin": "\*",

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

},

"body": "{\"message\":\"OTP sent to email\"}"

}

Sample response for successful OTP validation:

{

"statusCode": 200,

"headers": {

"Access-Control-Allow-Origin": "\*",

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

},

"body": "{\"message\":\"OTP verified successfully\",\"statusCode\":200}"

}

### 5. Dependency

The API relies on the following dependencies:

* **nodemailer**: For sending emails.
* **mongoose**: For MongoDB database operations.
* **Google APIs (googleapis)**: For accessing Gmail OAuth2 for email sending.
* **MongoDB**: For storing and retrieving user data.

### 6. Logic

The function follows a logical flow:

* **User Verification**: Checks if the user email is registered in the database. If not, returns an error response.
* **OTP Generation**: Generates a random 6-digit OTP and sends it to the user's email. Stores the OTP in an in-memory cache for validation.
* **OTP Validation**: Retrieves the OTP from the cache and compares it with the OTP provided by the user. Returns a success or error response accordingly

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,

}

});

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

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

exports.handler = async (event) => {

try {

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

const lowerCaseEmail = email.toLowerCase();

console.log(lowerCaseEmail);

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

if (user) {

return {

statusCode: 404,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

body: JSON.stringify({ message: 'Email already registered', success: false }),

};

}

if (action === 'generate') {

// Generate OTP and send via email

const generatedOtp = await generateOtp(lowerCaseEmail);

return {

statusCode: 200,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

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

};

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

// Validate entered OTP

const isValid = validateOtp(lowerCaseEmail, otp);

if (isValid) {

return {

statusCode: 200,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

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

};

} else {

return {

statusCode: 400,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

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

};

}

} else {

return {

statusCode: 400,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

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

};

}

} catch (error) {

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

return {

statusCode: 500,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

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

},

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;

}

MODEL ->

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

email: { type: String, required: true, unique: true }

});

module.exports = mongoose.model('User2', userSchema);