**API’s LAMBDA FUN NAME -> forgetpasswordLN (login side)**

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

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

Stage name -> users

**1. Role of the API**

This API handles the generation and validation of One-Time Passwords (OTPs) for user authentication. It sends an OTP to a registered user's email and validates the OTP entered by the user.

**2. Functioning**

The function processes POST requests with a specified action (**generate** or **validate**) and user details. It verifies the existence of the user in the database, generates an OTP, sends it to the user's email, or validates an entered OTP against a 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\": \"sainips941466@gmail.com\", \"action\": \"generate\"}"

}

Example of a request body for validating an OTP: {

"body": "{\"email\": \"sainips941466@gmail.com\", \"action\": \"validate\", \"otp\": \"408215\"}"

}

### 4. Response

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

* **200 OK**: For successful OTP generation or validation.
* **404 Not Found**: If the user's email is not registered.
* **400 Bad Request**: For invalid actions or if the entered OTP is incorrect.
* **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\",\"statusCode\":200}"

}

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

Key components this API relies on:

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

### 6. Logic

The function follows a sequence based on the user action:

* **User Verification**: Checks if the user is registered by querying the MongoDB database.
* **OTP Generation**:
  + Generates a random 6-digit OTP.
  + Sends this OTP to the user's email.
  + Stores the OTP temporarily in an in-memory cache for subsequent validation.
* **OTP Validation**:
  + Retrieves the OTP from the cache and compares it with the OTP provided by the user.
  + Returns a success message if they match, otherwise an error message.

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();

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: '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': '\*', // Allow requests from any origin

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

},

body: JSON.stringify({ message: 'OTP sent to email', statusCode:200 }),

};

} 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', statusCode:200 }),

};

} 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' }),

};

}

} 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);