### LAMBDA FUNCTION -> **RegNEW**

API URL -> <https://7efwp1v3ed.execute-api.us-east-1.amazonaws.com/check/register>

**(DashbordAPI’s)**

**Resource -> register**

**Stage -> check**

### 1. Role of API

The API is designed to handle user registration by creating a new user in the MongoDB database. It validates the request data, checks for existing users, and returns appropriate responses.

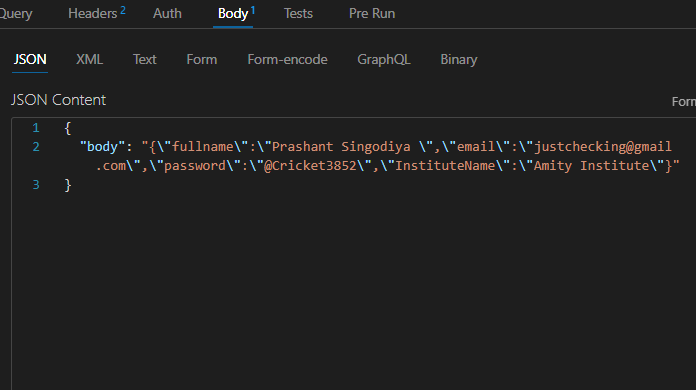
### 2. Functioning

This AWS Lambda function connects to a MongoDB database, validates the incoming request, and then either creates a new user or returns an error response if validation fails or if the user already exists.

### 3. Request Body

The request body must be a JSON object with the following fields:

* fullname (String, required): The full name of the user.
* email (String, required): The email address of the user.
* password (String, required): The password for the user account.
* InstituteName (String, optional): The name of the institute associated with the user.



### 4. Response

The API provides the following responses:

* **201 Created**: The user is successfully registered.

A screen shot of a computer screen

Description automatically generated

**400 Bad Request**: The request is missing required fields or contains invalid data.

**500 Internal Server Error**: An error occurred on the server while processing the request.

json

### 5. Logic

1. **Connect to MongoDB**: The function connects to the MongoDB database using the URI stored in the environment variable MONGO\_URI.
2. **Parse Request Body**: The request body is parsed to extract the fullname, email, password, and InstituteName fields.
3. **Validate Fields**: The function checks if the fullname, email, and password fields are provided. If any are missing, it returns a 400 error.
4. **Validate Email Format**: The function uses a regex to validate the email format. If invalid, it returns a 400 error.
5. **Check Existing Email**: The function checks if a user with the same email already exists. If yes, it returns a 400 error.
6. **Create and Save User**: If all validations pass, a new user is created and saved in the database.
7. **Return Success Response**: If the user is successfully created, the function returns a 201 status code with a success message.
8. **Handle Errors**: If any errors occur during processing, a 500 error response is returned.
9. **Disconnect from MongoDB**: Finally, the function ensures it disconnects from the MongoDB database.

### 6. Dependencies

* **AWS Lambda**: The function is designed to run as an AWS Lambda function.
* **MongoDB**: The function interacts with a MongoDB database to store user information.
* **Mongoose**: The function uses Mongoose for MongoDB object modeling and database interaction.
* **Node.js**: The function is written in Node.js and requires the appropriate Node.js runtime.

CODE ->

INDEX>JS

**const mongoose=require('mongoose');**

**const User=require('./model.js');**

**const mongoUri = process.env.MONGO\_URI;**

**exports.handler= async (event) => {**

**let client;**

**try {**

**client=await mongoose.connect(mongoUri);**

**const {fullname, email, password, InstituteName}= JSON.parse(event.body);**

**if(!fullname || !email || !password) {**

**return{**

**statusCode: 400,**

**headers: {**

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

**"Access-Control-Allow-Headers": "Content-Type",**

**'Access-Control-Allow-Credentials': true,**

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

**},**

**body:JSON.stringify({message: 'Missing fields'}),**

**};**

**}**

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

**if (!emailRegex.test(email)) {**

**return {**

**statusCode: 400,**

**headers: {**

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

**"Access-Control-Allow-Headers": "Content-Type",**

**'Access-Control-Allow-Credentials': true,**

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

**},**

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

**};**

**}**

**const existingEmail = await User.findOne({ email });**

**if (existingEmail) {**

**return {**

**statusCode: 400,**

**headers: {**

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

**"Access-Control-Allow-Headers": "Content-Type",**

**'Access-Control-Allow-Credentials': true,**

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

**},**

**body: JSON.stringify({ message: 'Email already exists' }),**

**};**

**}**

**const user=new User({**

**fullname,**

**email,**

**// contact,**

**InstituteName,**

**password,**

**});**

**await user.save();**

**return {**

**statusCode: 201,**

**headers: {**

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

**"Access-Control-Allow-Headers": "Content-Type",**

**'Access-Control-Allow-Credentials': true,**

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

**},**

**body: JSON.stringify({ message: 'User registered successfully', statusCode:201 }),**

**};**

**} catch (error) {**

**console.error('Registration error:', error);**

**return {**

**statusCode: 500,**

**headers: {**

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

**"Access-Control-Allow-Headers": "Content-Type",**

**'Access-Control-Allow-Credentials': true,**

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

**},**

**body: JSON.stringify({ message: 'Failed to register user' }),**

**};**

**} finally {**

**if (client) {**

**await client.disconnect(); // Disconnect from MongoDB**

**}**

**}**

**};**

**MODEL ->**

A screenshot of a computer program

Description automatically generated