### LAMBDA FUNCTION -> **userProfile**

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

Resource -> **DashbordAPI's ->** [**P\_details**](https://us-east-1.console.aws.amazon.com/apigateway/main/apis/7efwp1v3ed/resources?api=7efwp1v3ed&region=us-east-1)

Stage -> profile

### 1. Role of the API

This API fetches user details from the database based on the provided email, excluding the password field.

### 2. Functioning

* **Token Verification**: Validates the JWT token to authenticate the request.
* **CORS Handling**: Sets appropriate headers to handle Cross-Origin Resource Sharing (CORS).
* **Database Connection**: Ensures a consistent MongoDB connection.
* **Request Validation**: Ensures required fields like email are present and valid.
* **User Retrieval**: Fetches user details by email from the database.
* **Response Handling**: Returns success or error messages based on the operation's outcome.

### 3. Request Body

**Required:**

{

"headers": {

"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMTk3OTA4LCJleHAiOjE3MjEyMTU5MDh9.WxVBGrx8W\_BY2UCaVGwNpth3qux6qA81V\_e-bA2PNKQ"

},

"body": "{\"email\": \"sainips941466@gmail.com\"}"

}

**// PLEASE NOTE - > API will Take 2 payload, On first Call it wil takes – Email And on Second call it will take Email & UserProfileLink (it will generated from s3 bucket- and only when user want to set their DP) then we call again with these two payload.**

### 4. Response

**Success Response:**

* **Status Code**: 200
* **Body**: JSON representation of the user object (excluding password)

{

"statusCode": 200,

"headers": {

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

"Access-Control-Allow-Credentials": true

},

"body": {

"\_id": "6671ae556a5f4a4c5ca333b5",

"fullname": "Prashant Singodiya1",

"email": "sainips941466@gmail.com",

"InstituteName": "Amity Nodia",

"createdAt": "2024-06-18T15:57:09.389Z",

"\_\_v": 0,

"UserProfileLink": "https://exambuilder-alfabeto.s3.amazonaws.com/profile/1719509020747\_image.jpg"

}

}

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Unauthorized"}
* **Status Code**: 400
  + **Body**: {"error": "Email is required"}
* **Status Code**: 404
  + **Body**: {"error": "User not found"}
* **Status Code**: 500
  + **Body**: {"error": "Internal Server Error"}

### 5. Logic

1. **Token Verification**: Validates the JWT token to ensure the user is authorized to perform the operation.
2. **CORS Headers**: Sets headers to allow CORS requests from any origin.
3. **Database Connection**: Ensures a consistent connection to MongoDB using Mongoose.
4. **Request Validation**: Ensures required fields like email are present and valid.
5. **User Retrieval**: Fetches user details by email from the database.
6. **Response Creation**: Generates and returns appropriate responses based on the success or failure of the operation.

### 6. Dependencies

* **mongoose**: MongoDB object modeling tool.
* **jsonwebtoken**: Library for generating and verifying JSON Web Tokens (JWT).

CODE ->

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const User2 = require('./model.js');

const MONGODB\_URI = process.env.MONGO\_URI;

const JWT\_SECRET = process.env.JWT\_SCREAT;

let conn = null;

// Function to connect to MongoDB

async function connectToDatabase() {

if (conn == null) {

conn = await mongoose.connect(MONGODB\_URI);

}

return conn;

}

// Lambda handler

exports.handler = async (event) => {

// Enable CORS (Cross-Origin Resource Sharing)

const headers = {

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

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

};

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

// Handle CORS preflight requests

return {

statusCode: 200,

headers: {

...headers,

'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',

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

},

body: JSON.stringify({ message: 'Preflight check passed' }),

};

}

const token = event.headers.Authorization || event.headers.authorization;

if (!token) {

return {

statusCode: 401,

headers: headers,

body: JSON.stringify({ error: 'Unauthorized' }),

};

}

try {

// Verify JWT token

const decoded = jwt.verify(token, JWT\_SECRET);

// Parse the request body to get the email and optionally UserProfileLink

const requestBody = JSON.parse(event.body);

const { email, UserProfileLink } = requestBody;

if (!email) {

return {

statusCode: 400,

headers: headers,

body: JSON.stringify({ error: 'Email is required' }),

};

}

// Connect to MongoDB

await connectToDatabase();

// Fetch user details by email

let user = await User2.findOne({ email: email.toLowerCase() }).select('-password');

if (!user) {

return {

statusCode: 404,

headers: headers,

body: JSON.stringify({ error: 'User not found' }),

};

}

// Update UserProfileLink if provided

if (UserProfileLink) {

user.UserProfileLink = UserProfileLink;

await user.save();

}

// Return updated user data

user = user.toObject(); // Convert mongoose document to plain JavaScript object

delete user.password; // Ensure password is not included in the response

return {

statusCode: 200,

headers: headers,

body: user

};

} catch (error) {

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

// Handle errors

return {

statusCode: 500,

headers: headers,

body: JSON.stringify({ error: 'Internal Server Error' }),

};

}

};

MODEL ->

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

fullname: { type: String, required: true, unique: false },

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

// contact: { type: String, required: false, unique: true },

password: { type: String, required: true, unique: false },

InstituteName: { type: String, required: true, unique: false },

UserProfileLink: {type: String, require:false, unique: false},

createdAt: { type: Date, default: Date.now }

});

// Middleware to convert email to lowercase before saving

userSchema.pre('save', function(next) {

const user = this;

// Convert email to lowercase (if it exists and is modified)

if (user.email && user.isModified('email')) {

user.email = user.email.toLowerCase();

}

// Call next to proceed with the save operation

next();

});

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