### LAMBDA FUNCTION name \_> **userloginN**

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

DashbordAPI’s

Resource -> login

Stage-> authcheck

### 1. Role of API

This API handles user login by validating user credentials (email and password). It returns a JWT token if the login is successful, allowing the user to authenticate future requests.

### 2. Functioning

The AWS Lambda function connects to a MongoDB database, validates the incoming request, checks the user's credentials, and generates a JWT token if the credentials are valid.

### 3. Request Body

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

* email (String, required): The email address of the user.
* password (String, required): The password for the user account.

A screen shot of a computer

Description automatically generated

### 4. Response

The API provides the following responses:

* **200 OK**: The login is successful, and a JWT token is returned.

A computer screen shot of text

Description automatically generated

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

**401 Unauthorized**: The password is incorrect.

**404 Not Found**: The user with the provided email does not exist.

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

### 5. Logic

1. **Check Event Body**: The function first checks if the request body is empty. If it is, a 400 error is returned.
2. **Parse Request Body**: The request body is parsed to extract the email and password fields.
3. **Convert Email to Lowercase**: The email is converted to lowercase to ensure case-insensitive comparison.
4. **Validate Fields**: The function checks if both email and password fields are provided. If either is missing, a 400 error is returned.
5. **Validate Email Format**: The function uses a regex to validate the email format. If invalid, a 400 error is returned.
6. **Find User by Email**: The function looks up the user in the database by the lowercase email.
7. **Check User Existence**: If the user does not exist, a 404 error is returned.
8. **Compare Password**: The function compares the provided password with the stored password. If they do not match, a 401 error is returned.
9. **Generate JWT Token**: If the password matches, a JWT token is generated with the user's ID and email.
10. **Return Success Response**: The function returns a 200 status code with the JWT token and the user's full name.
11. **Handle Errors**: If any errors occur during processing, a 500 error response is returned.

### Dependencies

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

CODE ->

const mongoose = require('mongoose');

//const crypto = require('crypto');

const jwt = require('jsonwebtoken');

const User = require('./model.js'); // Assuming your model file is in the same directory

// Mongoose connection setup

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

const db = mongoose.connection;

// Handle connection errors

db.on('error', console.error.bind(console, 'MongoDB connection error:'));

exports.handler = async (event) => {

// Check if the event body is empty

if (!event.body) {

return {

statusCode: 400,

headers: {

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

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

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

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

},

body: JSON.stringify({ message: "Empty request body." })

};

}

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

// Convert email to lowercase

const lowercaseEmail = email.toLowerCase();

if (!lowercaseEmail || !password) {

return {

statusCode: 400,

headers: {

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

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

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

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

},

body: JSON.stringify({ message: "Email and password are required." })

};

}

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

if (!emailRegex.test(lowercaseEmail)) {

return {

statusCode: 400,

headers: {

'Access-Control-Allow-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' }),

};

}

try {

// Find user by lowercase email

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

if (!user) {

return {

statusCode: 404,

headers: {

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

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

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

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

},

body: JSON.stringify({ message: "User not found." })

};

}

// Compare password

if (password !== user.password) {

return {

statusCode: 401,

headers: {

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

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

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

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

},

body: JSON.stringify({ message: "Invalid password." })

};

}

// Create JWT token with payload including user's email

const tokenPayload = {

sub: user.\_id,

email: lowercaseEmail

};

const token = jwt.sign(tokenPayload, process.env.JWT\_SCREAT, { expiresIn: '5h' });

// Return success response with JWT token

return {

statusCode: 200,

headers: {

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

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

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

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

},

body: JSON.stringify({

message: "Login successful!",

token: token,

fullName: user.fullName, // Assuming the field is fullName

}),

};

} catch (error) {

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

return {

statusCode: 500,

headers: {

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

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

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

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

},

body: JSON.stringify({ message: "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 },**

**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);**