**Lambda Function Name**: userLoginN\_R

**API Name**: Module1\_Refactored

**Resource Name**: userLoginN\_R

**API URL**: https://aymk6mhnl3.execute-api.us-east-1.amazonaws.com/v1/userLoginN\_R

1. **Role of the API**  
   This API serves as an authentication endpoint for user login. It validates user credentials stored in MongoDB and provides a JSON Web Token (JWT) for authorized access to protected resources.
2. **Functioning**  
   Upon receiving a POST request with user credentials, the API validates the email format and checks if the user exists in the database. It then verifies the provided password against the stored password hash. If successful, it returns a JWT that can be used for subsequent authenticated requests.
3. **Request Body**  
   The API expects a JSON object in the POST request body with the following fields:
   * **email**: User's email address.
   * **password**: User's password.

Example:

json

Copy code

{

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

"password": "5478674eef03bb5cf47e7c3216ce2f01"

}

1. **Response**  
   The API responds with the following status codes and JSON body:
   * **200 OK**: Successful login, returns a JWT.
   * **400 Bad Request**: Missing or invalid email or password format.
   * **401 Unauthorized**: Incorrect password.
   * **404 Not Found**: User not found in the database.
   * **500 Internal Server Error**: Unexpected errors during processing.

Example response body for successful login:

json

Copy code

{

"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": "{\"message\":\"Login successful!\",\"token\":\"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjJiZGI5ODNmYmI0NjE3ZTg5NzIwZWMiLCJpYXQiOjE3MTQzNzEyMjcsImV4cCI6MTcxNDM3NDgyN30.jr1Tj7OrfwJNjWY9k5DHLKoK2WJNYhozqPJdzBvHQsE\"}"

}

1. **Dependency**  
   The API relies on:
   * **mongoose**: For MongoDB interactions.
   * **jsonwebtoken**: For JWT generation.
   * **MongoDB**: Database for storing and validating user credentials.
2. **Logic**
   * **Email and Password Validation**: Validates the email format and checks for presence of email and password.
   * **User Authentication**: Queries MongoDB for user with provided email, compares passwords.
   * **Token Generation**: Generates JWT upon successful authentication.
   * **Error Handling**: Handles errors like missing user, incorrect password, or database connection issues.

CODE:

const mongoose = require('mongoose');

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:'));

// Common CORS headers for preflight

const commonHeaders = {

'Access-Control-Allow-Origin': 'https://admin.exambuilder.online',

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

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

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

};

exports.handler = async (event) => {

// Handle preflight (OPTIONS) request

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

return {

statusCode: 204,

headers: commonHeaders,

body: null

};

}

// CORS header for other requests

const responseHeaders = {

'Access-Control-Allow-Origin': 'https://admin.exambuilder.online'

};

// Check if the event body is empty

if (!event.body) {

return {

statusCode: 400,

headers: responseHeaders,

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

};

}

let parsedBody;

try {

parsedBody = JSON.parse(event.body);

} catch (error) {

return {

statusCode: 400,

headers: responseHeaders,

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

};

}

const { email, password } = parsedBody;

// Convert email to lowercase

const lowercaseEmail = email ? email.toLowerCase() : '';

if (!lowercaseEmail || !password) {

return {

statusCode: 400,

headers: responseHeaders,

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

};

}

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

if (!emailRegex.test(lowercaseEmail)) {

return {

statusCode: 400,

headers: responseHeaders,

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: responseHeaders,

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

};

}

// Compare password

if (password !== user.password) {

return {

statusCode: 401,

headers: responseHeaders,

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\_SECRET, { expiresIn: '5h' });

// Return success response with JWT token

return {

statusCode: 200,

headers: responseHeaders,

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: responseHeaders,

body: JSON.stringify({ message: "Internal server error" })

};

}

};