### LAMBDA FUNCTION -> **dashbordEdit\_R**

API URL -> https://j0mmgihtaj.execute-api.us-east-1.amazonaws.com/v1/dashbordEdit\_R

Resource -> **dashbordEdit\_R**

Stage -> v1

### 1. Role of the API

This API handles updating a quiz's title and completion status in the database.

### 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.
* **Quiz Update**: Updates the quiz title and/or completion status based on the provided \_id.
* **Response Handling**: Returns success or error messages based on the operation's outcome.

### 3. Request Body

**Required:**

Headers:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIyMDA3ODg1LCJleHAiOjE3MjIwMjU4ODV9.oDt80mp8EQb3uf73lZ-j2789DEOZ8dtxsdbOP5z1Z6g

Content-Type: application/json

Body:

{

"\_id": "66704b60acc1addf43d8d253",

"quizTitle": "computer",

"isCompleted": true

}

quizTitle -> Optional

isCompleted -> Optional

### 4. Response

**Success Response:**

* **Status Code**: 200

{

"statusCode": 200,

"headers": {

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

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

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

},

"body": "{\"message\":\"Data edited successfully\"}"

}

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Authorization token missing"}
  + **Body**: {"error": "Invalid or expired token"}
* **Status Code**: 400
  + **Body**: {"error": "Invalid request payload"}
* **Status Code**: 404
  + **Body**: {"error": "Quiz 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 \_id are present and valid.
5. **Quiz Update**: Updates the quiz title and/or completion status based on the provided \_id.
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 Quiz = require('./question.js'); // Ensure this path is correct for your Quiz model

// Environment Variables

const uri = process.env.MONGODB\_URI; // MongoDB connection string from environment variable

const jwtSecret = process.env.JWT\_SECRET\_KEY; // JWT secret key from environment variable

let cachedDb = null;

// Function to connect to MongoDB

async function connectToDatabase() {

if (cachedDb && mongoose.connection.readyState === 1) {

return cachedDb;

}

try {

cachedDb = await mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology: true });

console.log('MongoDB connected');

return cachedDb;

} catch (error) {

console.error('MongoDB connection error:', error);

throw new Error('Could not connect to the database');

}

}

// JWT verification function

function verifyToken(token) {

try {

return jwt.verify(token, jwtSecret);

} catch (error) {

throw new Error('Invalid or expired token');

}

}

exports.handler = async (event) => {

// CORS headers

const headers = {

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

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

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

};

// Handle CORS preflight request

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

return {

statusCode: 200,

headers,

body: JSON.stringify({ message: 'CORS preflight' })

};

}

// Add Access-Control-Allow-Origin header for non-preflight requests

const responseHeaders = {

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

};

try {

// Extract JWT token from headers and validate

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

if (!authHeader) {

return {

statusCode: 401,

headers: responseHeaders,

body: JSON.stringify({ error: 'Authorization token missing' }),

};

}

const token = authHeader.replace('Bearer ', '');

if (!token) {

return {

statusCode: 401,

headers: responseHeaders,

body: JSON.stringify({ error: 'Authorization token missing' }),

};

}

// Verify JWT token

const decoded = verifyToken(token);

// Extract and validate request body

let requestBody;

try {

requestBody = JSON.parse(event.body);

} catch (jsonError) {

return {

statusCode: 400,

headers: responseHeaders,

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

};

}

const { \_id, quizTitle, isCompleted } = requestBody;

// Validate required fields

if (!\_id || (quizTitle === undefined && isCompleted === undefined)) {

return {

statusCode: 400,

headers: responseHeaders,

body: JSON.stringify({ error: 'Invalid request payload' }),

};

}

// Connect to MongoDB

await connectToDatabase();

// Find the quiz by ID

const quiz = await Quiz.findById(\_id);

if (!quiz) {

return {

statusCode: 404,

headers: responseHeaders,

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

};

}

// Update quiz only if necessary

let updated = false;

if (quizTitle !== undefined && quiz.quizTitle !== quizTitle) {

quiz.quizTitle = quizTitle;

updated = true;

}

if (isCompleted !== undefined && quiz.isCompleted !== isCompleted) {

quiz.isCompleted = isCompleted;

updated = true;

}

// Save the updated quiz

if (updated) {

await quiz.save();

}

return {

statusCode: 200,

headers: responseHeaders,

body: JSON.stringify({ message: 'Data edited successfully' })

};

} catch (error) {

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

if (error.message === 'Invalid or expired token') {

return {

statusCode: 401,

headers: responseHeaders,

body: JSON.stringify({ error: 'Invalid or expired token' }),

};

} else if (error.message === 'Could not connect to the database') {

return {

statusCode: 500,

headers: responseHeaders,

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

};

} else {

return {

statusCode: 500,

headers: responseHeaders,

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

};

}

}

};