### LAMBDA FUNCTION -> **DashbordDel**

API URL ->

<https://7efwp1v3ed.execute-api.us-east-1.amazonaws.com/dashbord/dashDEL>

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

Stage -> dashbord

### 1. Role of the API

This API handles the update of a quiz's status to false 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 status to false based on the provided \_id.
* **Response Handling**: Returns success or error messages based on the operation's outcome.

### 3. Request Body

**Required:**

json

{

"headers": {

"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMTk0NTEyLCJleHAiOjE3MjEyMTI1MTJ9.L1xBwgXYsXitQrcPy8mok2OT5E\_\_UfWzFM4WH63yuAI"

},

"body": "{\"\_id\":\"6695096e94643cb35db3d63b\"}"

}

### 4. Response

**Success Response:**

* **Status Code**: 200

{

"statusCode": 200,

"headers": {

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

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

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

},

"body": "{\"message\":\"Quiz status updated to false successfully\"}"

}

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"message": "No token provided"}
  + **Body**: {"message": "Invalid token"}
* **Status Code**: 400
  + **Body**: {"message": "Quiz ID is required"}
  + **Body**: {"message": "Invalid Quiz ID format"}
* **Status Code**: 404
  + **Body**: {"message": "Quiz not found"}
* **Status Code**: 200
  + **Body**: {"message": "Quiz status is already false"}
* **Status Code**: 500
  + **Body**: {"message": "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 status to false 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'); // Adjust the path as needed

const MONGO\_URI = process.env.MONGODB\_URI; // Add your MongoDB connection string

const JWT\_SECRET = process.env.JWT\_SECRET\_KEY; // Add your JWT secret

let isConnected = false;

const connectToDatabase = async () => {

if (isConnected) {

return;

}

await mongoose.connect(MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

});

isConnected = true;

};

exports.handler = async (event) => {

// Handle CORS preflight request

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

return {

statusCode: 200,

headers: {

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

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

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

},

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

};

}

try {

await connectToDatabase();

const token = event.headers.Authorization;

if (!token) {

return {

statusCode: 401,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'No token provided' }),

};

}

let decoded;

try {

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

} catch (err) {

return {

statusCode: 401,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'Invalid token' }),

};

}

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

const quizId = body.\_id;

if (!quizId) {

return {

statusCode: 400,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'Quiz ID is required' }),

};

}

// Validate quizId format

if (!mongoose.Types.ObjectId.isValid(quizId)) {

return {

statusCode: 400,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'Invalid Quiz ID format' }),

};

}

const quiz = await Quiz.findById(quizId).exec();

if (!quiz) {

return {

statusCode: 404,

headers: {

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

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

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

},

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

};

}

if (quiz.status === false) {

return {

statusCode: 200,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'Quiz status is already false' }),

};

}

quiz.status = false;

await quiz.save();

return {

statusCode: 200,

headers: {

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

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

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

},

body: JSON.stringify({ message: 'Quiz status updated to false successfully' }),

};

} catch (err) {

console.error(err);

return {

statusCode: 500,

headers: {

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

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

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

},

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

};

}

};

MODEL -> question.js

const mongoose = require('mongoose');

// Define schema for multiple choice questions

const MCQSchema = new mongoose.Schema({

question: {

type: String,

required: false,

},

questionImageLink: {

type: String,

required: false,

},

options: [{

answer: {

type: String,

required: false,

},

answerImageLink: {

type: String,

required: false,

}

}],

correctAnswer: {

type: String,

required: true,

},

description: {

type: String,

required: false

},

version: {

type: Number,

default: 1

}

});

// Define schema for descriptive questions

const DescriptiveSchema = new mongoose.Schema({

question: {

type: String,

required: false,

},

questionImageLink: {

type: String,

required: false,

},

answer: {

type: String,

required: false

},

answerImageLink: {

type: String,

required: false,

},

version: {

type: Number,

default: 1

}

});

// Define main quiz schema

const QuizSchema = new mongoose.Schema({

quizTitle: {

type: String,

required: true,

},

creatorName: {

type: String,

required: false

},

creatorEmail: {

type: String,

required: true

},

isCompleted: {

type: Boolean,

required: false,

default: false

},

status: {

type: Boolean,

required: false,

default: true

},

preVersionID: {

type: [mongoose.Schema.Types.ObjectId],

default: []

},

mcqQuizz: [MCQSchema], // Array of multiple choice questions

descriptiveQuizz: [DescriptiveSchema], // Array of descriptive questions

quizDuration: {

type: Number,

required: false

}

}, { timestamps: { createdAt: 'createdAt' } });

// Pre-save middleware to set creatorName from User2's fullname

QuizSchema.pre('save', async function(next) {

if (this.isNew || this.isModified('creatorEmail')) {

const User2 = mongoose.model('User2');

const user = await User2.findOne({ email: this.creatorEmail }).exec();

if (user) {

this.creatorName = user.fullname;

} else {

const error = new Error('User not found');

error.statusCode = 404;

return next(error);

}

}

next();

});

// Middleware to handle versioning of questions in mcqQuizz and descriptiveQuizz

QuizSchema.pre('save', async function(next) {

if (!this.isNew) {

const originalQuiz = await mongoose.model('Quiz').findById(this.\_id).exec();

if (originalQuiz) {

this.mcqQuizz.forEach(mcq => {

const originalMCQ = originalQuiz.mcqQuizz.id(mcq.\_id);

if (originalMCQ && !mcq.\_id.equals(originalMCQ.\_id)) {

mcq.version = originalMCQ.version + 1;

}

});

this.descriptiveQuizz.forEach(dq => {

const originalDQ = originalQuiz.descriptiveQuizz.id(dq.\_id);

if (originalDQ && !dq.\_id.equals(originalDQ.\_id)) {

dq.version = originalDQ.version + 1;

}

});

// Handle deleted questions by adding their IDs to preVersionID

originalQuiz.mcqQuizz.forEach(originalMCQ => {

if (!this.mcqQuizz.id(originalMCQ.\_id)) {

this.preVersionID.push(originalMCQ.\_id);

}

});

originalQuiz.descriptiveQuizz.forEach(originalDQ => {

if (!this.descriptiveQuizz.id(originalDQ.\_id)) {

this.preVersionID.push(originalDQ.\_id);

}

});

}

}

next();

});

// Create a model using the schema

const Quiz = mongoose.model('Quiz', QuizSchema);

module.exports = Quiz;