**API’s LAMBDA FUN NAME -> quizDuration**

**API’s URL ->** [**https://r9qkgroo29.execute-api.us-east-1.amazonaws.com/v1/duration**](https://r9qkgroo29.execute-api.us-east-1.amazonaws.com/v1/duration)

**API name: quizDuration**

**Resource: duration**

1. **Role of the API** This API updates the duration of a quiz identified by its ID in a MongoDB database. It verifies the validity of a JWT token provided in the Authorization header using the Bearer scheme to authenticate and authorize the update operation.
2. **Functioning** The function handles POST requests containing the quiz ID and quiz duration in the request body. It expects a JWT token in the Authorization header for authentication. Upon successful validation, it updates the specified quiz's duration in the database and returns a response with the updated quiz ID and duration.
3. **Request Body** The API expects a JSON object with the following fields:
   * quizId: The ID of the quiz to update.
   * quizDuration: The new duration of the quiz in minutes.

Example of a valid request body:

**Body:**

{

"quizId": "615d7f5c3e0b3a001fd434a2",

"quizDuration": 30

}

**Header:**

Authorization: your\_jwt\_token\_here

1. The API provides the following responses:
   * 200 OK: Quiz duration updated successfully. Returns the updated quizId and quizDuration.
   * 400 Bad Request: Missing quiz ID, quiz duration, or token.
   * 401 Unauthorized: Invalid or missing JWT token.
   * 404 Not Found: Quiz with the specified ID not found in the database.
   * 500 Internal Server Error: Issues during the update process or database connection.
2. **Dependency** This API relies on the following components:
   * mongoose: For handling MongoDB operations.
   * jsonwebtoken: For JWT token verification.
   * Quiz model (quizSchema.js): The mongoose model that defines the schema for quiz data.
   * MongoDB: The database where quiz records are stored.
3. **Logic**
   * **Token Validation**: The API verifies the JWT token provided in the Authorization header using the Bearer scheme to authenticate and authorize the update operation.
   * **Database Update**: It updates the quiz duration for the specified quiz ID using findByIdAndUpdate method of Mongoose.
   * **Error Handling**: Handles various error scenarios such as missing fields, invalid tokens, quiz not found, and database errors.
   * **Resource Management**: Ensures proper closure of database connections in the finally block to manage resources efficiently.

CODE ->

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const Quiz = require('./question.js'); // Ensure the path to your schema file is correct

const JWT\_SECRET = process.env.JWT\_SECRET; // Ensure you set this environment variable

exports.handler = async (event) => {

console.log('Event:', event); // Debug log to see the entire event

// Common CORS headers

const corsHeaders = {

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

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

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

};

// Handle preflight requests

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

return {

statusCode: 200,

headers: corsHeaders,

body: ''

};

}

// Extract headers and body

const headers = event.headers;

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

console.log('Headers:', headers);

console.log('Body:', body);

const { quizId, quizDuration } = body;

let token = null;

// Check Authorization header

const authorizationHeader = headers.Authorization || headers.authorization;

if (authorizationHeader) {

const tokenParts = authorizationHeader.split(' ');

if (tokenParts.length === 2 && tokenParts[0].toLowerCase() === 'bearer') {

token = tokenParts[1];

} else {

token = authorizationHeader; // Handle case without "Bearer " prefix

}

}

console.log('Token:', token);

// Validate input

if (!quizId || !quizDuration || !token) {

return {

statusCode: 400,

headers: corsHeaders,

body: JSON.stringify({ message: "Quiz ID, quiz duration, and token are required" })

};

}

try {

// Verify the JWT token

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

console.log('Decoded Token:', decoded);

// Connect to MongoDB

await mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

});

// Find the quiz by ID and update the duration

const quiz = await Quiz.findByIdAndUpdate(quizId, { quizDuration }, { new: true });

if (!quiz) {

return {

statusCode: 404,

headers: corsHeaders,

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

};

}

// Construct the response with only quizId and quizDuration

const response = {

quizId: quiz.\_id,

quizDuration: quiz.quizDuration

};

return {

statusCode: 200,

headers: corsHeaders,

body: JSON.stringify(response)

};

} catch (error) {

if (error.name === 'JsonWebTokenError') {

return {

statusCode: 401,

headers: corsHeaders,

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

};

}

return {

statusCode: 500,

headers: corsHeaders,

body: JSON.stringify({ message: error.message })

};

} finally {

// Close the database connection

mongoose.connection.close();

}

};

MODLE -> 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;