### LAMBDA FUNCTION -> **quizzDiscriptiveQuestionService**

API URL -> <https://ee4pmf8ys1.execute-api.us-east-1.amazonaws.com/questionstyle/descriptiveQuestion>

Resource -> **quizzQuestionService  ->** descriptiveQuestion

Stage -> questionstyle

### 1. Role of the API

The primary role of this API is to process and save descriptive questions from a provided payload into an existing quiz document in a MongoDB database.

### 2. Functioning

* **Authentication**: Validates the JWT token from the request header to ensure the user is authorized.
* **CORS Handling**: Handles Cross-Origin Resource Sharing (CORS) preflight requests.
* **Database Connection**: Connects to MongoDB using Mongoose, with caching to optimize performance.
* **Request Processing**: Extracts descriptive questions from the request body, validates the data, and appends it to an existing quiz document.
* **Response Handling**: Returns appropriate responses based on the success or failure of the operations.

**3. Request Body**

{

"headers": {

"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMTMwODY2LCJleHAiOjE3MjExNDg4NjZ9.5gpUxGLTlmt8ClRtwTv-ePX8sH2c6ScaTIrLc9bQhBM",

"Content-Type": "application/json"

},

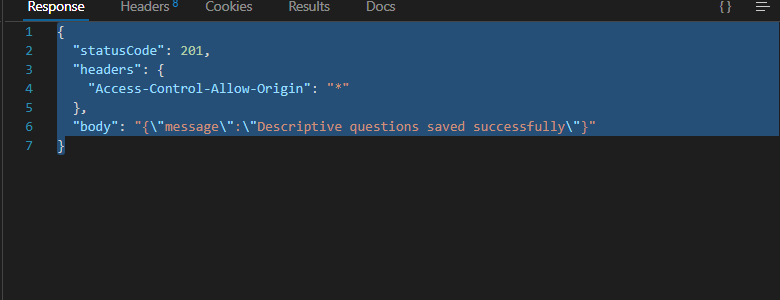
"body": "{\"quizTitle\":\"PrashantTesffft\",\"descriptiveQuizz\":[{\"question\":\"Describe the process of photosynthesis.\",\"questionImageLink\":\"www.google.com\",\"answer\":\"Photosynthesis is a process used by plants to convert light energy into chemical energy.\"}]}"

}

### 4. Response

**Success Response:**

* **Status Code**: 201



**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Unauthorized: No token provided"}
  + **Body**: {"error": "Unauthorized: Token expired or invalid"}
* **Status Code**: 400
  + **Body**: {"error": "Token does not contain an email"}
  + **Body**: {"error": "quizTitle and descriptiveQuizz are required fields and descriptiveQuizz should be an array"}
  + **Body**: {"error": "Each Descriptive Question object must have either question or questionImageLink defined"}
  + **Body**: {"error": "Each Descriptive Question object must have either answer or answerImageLink defined"}
  + **Body**: {"error": "Bad Request: <validation\_error\_message>"}

**Status Code**: 404

* **Body**: {"error": "Quiz with the provided title does not exist. Please provide a valid quiz title."}

**Status Code**: 500

* **Body**: {"error": "Internal Server Error: <error\_message>"}

### 5. Logic

1. **Token Verification**: Checks for the presence of the JWT token and verifies it. Returns an error if the token is missing or invalid.
2. **CORS Headers**: Sets headers to enable CORS and handle preflight requests.
3. **Database Connection**: Establishes a connection to MongoDB using Mongoose, with a caching mechanism to reuse the connection.
4. **Request Validation**: Extracts and validates the quizTitle and descriptiveQuizz from the request body. Ensures descriptiveQuizz is an array and contains valid data.
5. **Quiz Retrieval**: Searches for an existing quiz document with the provided quizTitle and creatorEmail.
6. **Descriptive Questions Validation**: Iterates through the descriptiveQuizz array and validates each question object, ensuring they have the required fields.
7. **Appending Descriptive Questions**: Appends the new descriptive question objects to the descriptiveQuizz array of the existing quiz document and saves it to MongoDB.
8. **Response Creation**: Creates and returns appropriate success or error responses based on the outcome of the operations.

### 6. Dependencies

* **mongoose**: For MongoDB object modeling.
* **jsonwebtoken**: For JWT token validation.
* **question.js**: Quiz schema model.

CODE ->

const mongoose = require('mongoose');

const Quiz = require('./question.js');

const jwt = require('jsonwebtoken');

const JWT\_SECRET\_KEY = process.env.JWT\_SECRET\_KEY;

let cachedDb = null;

const connectToDatabase = async () => {

if (cachedDb) {

return cachedDb;

}

try {

const connection = await mongoose.connect(process.env.MONGODB\_URI);

cachedDb = connection.connection.db;

return cachedDb;

} catch (error) {

throw new Error('Error connecting to MongoDB: ' + error.message);

}

};

exports.handler = async (event, context) => {

context.callbackWaitsForEmptyEventLoop = false;

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

return {

statusCode: 204,

headers: {

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

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

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

},

};

}

try {

await connectToDatabase();

const token = event.headers.Authorization;

if (!token) {

return createErrorResponse(401, 'Unauthorized: No token provided');

}

let decoded;

try {

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

} catch (error) {

return createErrorResponse(401, 'Unauthorized: Token expired or invalid');

}

if (!decoded.email) {

return createErrorResponse(400, 'Token does not contain an email');

}

const creatorEmail = decoded.email.toLowerCase();

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

const { quizTitle, descriptiveQuizz } = requestBody;

if (!quizTitle || !descriptiveQuizz || !Array.isArray(descriptiveQuizz)) {

return createErrorResponse(400, 'quizTitle and descriptiveQuizz are required fields and descriptiveQuizz should be an array');

}

// Validate Descriptive Question objects

for (let dq of descriptiveQuizz) {

if (!dq.question && !dq.questionImageLink) {

return createErrorResponse(400, 'Each Descriptive Question object must have either question or questionImageLink defined');

}

if (!dq.answer && !dq.answerImageLink) {

return createErrorResponse(400, 'Each Descriptive Question object must have either answer or answerImageLink defined');

}

}

let existingQuiz = await Quiz.findOne({ quizTitle, creatorEmail });

if (!existingQuiz) {

return createErrorResponse(404, 'Quiz with the provided title does not exist. Please provide a valid quiz title.');

}

// Merge descriptiveQuizz objects into existing quiz

existingQuiz.descriptiveQuizz = existingQuiz.descriptiveQuizz.concat(descriptiveQuizz);

await existingQuiz.save();

return createSuccessResponse(201, 'Descriptive questions saved successfully');

} catch (error) {

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

return createErrorResponse(400, 'Bad Request: ' + error.message);

}

return createErrorResponse(500, 'Internal Server Error: ' + error.message);

}

};

function createErrorResponse(statusCode, errorMessage) {

return {

statusCode,

headers: {

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

},

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

};

}

function createSuccessResponse(statusCode, message) {

return {

statusCode,

headers: {

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

},

body: JSON.stringify({ message })

};

}

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: false,

},

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

}, { 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;