### LAMBDA FUNCTION -> **quizzCoutMCQ**

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

Resource -> **quizzQuestionService ->** [**questionCount**](https://us-east-1.console.aws.amazon.com/apigateway/main/apis/ee4pmf8ys1/resources?api=ee4pmf8ys1&region=us-east-1)

Stage -> all

### 1. Role of the API

This API handles the retrieval of quiz information based on the quiz title and the creator's email, providing the total count of MCQ and descriptive quiz questions.

### 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 across Lambda invocations.
* **Quiz Retrieval**: Finds the quiz by the quiz title and the creator's email.
* **Count Calculation**: Calculates the total count of MCQ and descriptive quiz questions.
* **Response Handling**: Returns the total quiz count or appropriate error messages

**3. Request Body**

**{**

**"body": "{\"quizTitle\": \"AX\"}",**

**"headers": {**

**"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2Njk2NDMwNWNmOTFkZGVjNzVhN2JmMzUiLCJlbWFpbCI6Imp1c3RjaGVja2luZ0BnbWFpbC5jb20iLCJpYXQiOjE3MjExNzcwNjYsImV4cCI6MTcyMTE5NTA2Nn0.gVoyh7Jfzwjrj8a3SUIYDn\_Td3bAhPJjGpObf5\_6Ce4",**

**"Content-Type": "application/json"**

**}**

**}**

### 4. Response

**Success Response:**

* **Status Code**: 200

**{**

**"statusCode": 200,**

**"headers": {**

**"Content-Type": "application/json",**

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

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

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

**},**

**"body": "{\"quizTitle\":\"AX\",\"totalQuizzCount\":0}"**

**}**

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Authorization token is required"}
  + **Body**: {"error": "Token has expired"}
  + **Body**: {"error": "Invalid token"}
* **Status Code**: 400
  + **Body**: {"error": "quizTitle is required in the payload"}
* **Status Code**: 404
  + **Body**: {"error": "Quiz not found"}
* **Status Code**: 500
  + **Body**: {"error": "An error occurred while processing the request"}

### 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. **Quiz Retrieval**: Finds the quiz by the quiz title and the creator's email.
5. **Count Calculation**: Calculates the total number of questions (MCQ and descriptive) in the quiz.
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 Quiz = require('./question.js');**

**const jwt = require('jsonwebtoken');**

**// Ensure consistent MongoDB connection across Lambda invocations**

**let cachedDb = null;**

**const connectToDatabase = async () => {**

**if (cachedDb) {**

**return Promise.resolve(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);**

**}**

**};**

**const verifyToken = (token) => {**

**return new Promise((resolve, reject) => {**

**jwt.verify(token, process.env.JWT\_SECRET\_KEY, (err, decoded) => {**

**if (err) {**

**if (err.name === 'TokenExpiredError') {**

**reject(new Error('TokenExpiredError'));**

**} else {**

**reject(new Error('InvalidTokenError'));**

**}**

**} else {**

**resolve(decoded);**

**}**

**});**

**});**

**};**

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

**context.callbackWaitsForEmptyEventLoop = false; // Ensure Lambda doesn't wait for event loop to be empty**

**try {**

**// Check for Authorization header**

**if (!event.headers || (!event.headers.Authorization && !event.headers.authorization)) {**

**return {**

**statusCode: 401,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ error: 'Authorization token is required' })**

**};**

**}**

**const token = event.headers.Authorization || event.headers.authorization;**

**// Verify the token and extract email**

**const decodedToken = await verifyToken(token);**

**console.log(decodedToken);**

**const creatorEmail = decodedToken.email;**

**await connectToDatabase(); // Connect to MongoDB**

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

**const { quizTitle } = requestBody;**

**if (!quizTitle) {**

**return {**

**statusCode: 400,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ error: 'quizTitle is required in the payload' })**

**};**

**}**

**// Query MongoDB to find the quiz by quizTitle and email**

**const quiz = await Quiz.findOne({ quizTitle,creatorEmail });**

**if (!quiz) {**

**return {**

**statusCode: 404,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

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

**};**

**}**

**// Get the count of mcqQuizz and descriptiveQuizz objects for the found quiz**

**const mcqQuizzCount = quiz.mcqQuizz.length;**

**const descriptiveQuizzCount = quiz.descriptiveQuizz.length;**

**// Calculate the total quiz count**

**const totalQuizzCount = mcqQuizzCount + descriptiveQuizzCount;**

**// Return the counts as response**

**return {**

**statusCode: 200,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ quizTitle, totalQuizzCount })**

**};**

**} catch (error) {**

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

**// Handle specific JWT errors**

**if (error.message === 'TokenExpiredError') {**

**return {**

**statusCode: 401,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ error: 'Token has expired' })**

**};**

**} else if (error.message === 'InvalidTokenError') {**

**return {**

**statusCode: 401,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ error: 'Invalid token' })**

**};**

**}**

**return {**

**statusCode: 500,**

**headers: {**

**'Content-Type': 'application/json',**

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

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

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

**},**

**body: JSON.stringify({ error: 'An error occurred while processing the request' })**

**};**

**}**

**};**

**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;**