### LAMBDA FUNCTION NAME -> **quizzIdentity**

API URL -> <https://j0mmgihtaj.execute-api.us-east-1.amazonaws.com/v1/quizzIdentity>

Original API URL -> <https://ee4pmf8ys1.execute-api.us-east-1.amazonaws.com/info/quizzIdentity>

### 1. Role of the API

This API creates a new multiple-choice quiz (MCQ) for a user, ensuring that the quiz title is unique for the given user. It handles authentication via JWT tokens, maintains a consistent database connection, and processes request validation, quiz creation, and response formatting.

### 2. Functioning

* **Authentication**: The API uses JWT for authenticating requests. The token must be provided in the Authorization header.
* **Database Connection**: Establishes a connection to MongoDB using Mongoose, with a cached connection for efficiency.
* **Request Processing**: Parses the request body to extract the quiz title and MCQ questions, checks for existing quizzes with the same title, and creates a new quiz with a unique title if necessary.
* **Quiz Creation**: Saves the new quiz to the database.
* **Response**: Returns a success message with the new quiz title or appropriate error messages.

### 3. Request Body

The request body should be a JSON object containing the following fields:

* quizTitle: The title of the new quiz.
* mcqQuizz: The MCQ questions to be included in the quiz. (we extract email from JWT token)

Headers:

{

"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMDQ3NTA3LCJleHAiOjE3MjEwNjU1MDd9.8Et5CDThFtEuJsjQe-pgkbsEEx2aF1rPBNkyvlJwN\_c",

"Content-Type": "application/json"

}

Body:

{

"quizTitle": "PrashantTest",

"creatorName": "prashant",

"creatorEmail": "sainips941466@gmail.com"

}

### 4. Response with All Status Codes

* **201 Created**: Successfully created the quiz
* **400 Bad Request**: Missing or invalid input.
* **401 Unauthorized**: No token provided or token is invalid/expired
* **500 Internal Server Error**: An error occurred while processing the request

### 5. API Logic

1. **Authorization**: Verify the JWT token from the Authorization header.
2. **Database Connection**: Connect to MongoDB using Mongoose. If a connection already exists, use the cached connection.
3. **Request Validation**: Parse and validate the request body to ensure quizTitle and mcqQuizz are provided.
4. **Quiz Title Uniqueness**: Check if a quiz with the same title exists for the user. If it does, append a numeric suffix to the title to make it unique.
5. **Quiz Creation**: Create a new quiz document with the unique title and provided questions, and save it to the database.
6. **Response**: Return a success message with the new quiz title or appropriate error messages if any errors occur.

### 6. Dependency

* **Environment Variables**:
  + MONGODB\_URI: MongoDB connection URI.
  + JWT\_SECRET\_KEY: Secret key used for signing JWT tokens.
* **Packages**:
  + mongoose: For connecting to MongoDB.
  + jsonwebtoken: For handling JWT token verification

CODE:  
  
//updated code

const mongoose = require('mongoose');

const Quiz = require('./question.js'); // Adjust the path as necessary

const User2 = require('./User'); // Adjust the path as necessary

const jwt = require('jsonwebtoken');

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

// 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);

}

};

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

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

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

// Handle preflight requests

return {

statusCode: 204,

headers: {

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

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

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

},

};

}

try {

await connectToDatabase(); // Connect to MongoDB

const token = event.headers.Authorization;

let decoded;

try {

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

// If token is valid, proceed with processing the request

} catch (error) {

console.error('Error verifying token:', error);

return {

statusCode: 401, // Unauthorized

headers: {

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

},

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

};

}

// Ensure the token contains an email

if (!decoded.email) {

console.error('Token does not contain an email');

return {

statusCode: 400, // Bad Request

headers: {

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

},

body: JSON.stringify({ error: 'Bad Request: Token does not contain an email' }),

};

}

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

let { quizTitle, mcqQuizz } = requestBody;

const creatorEmail = decoded.email.toLowerCase();

// Check if a quiz with the same title exists for the user

let newQuizTitle = quizTitle;

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

let suffix = 1;

while (existingQuiz) {

newQuizTitle = `${quizTitle}${suffix}`;

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

suffix++;

}

// Create a new quiz instance with provided data

const newQuiz = new Quiz({

quizTitle: newQuizTitle,

creatorEmail, // Use the lowercase email for storing

mcqQuizz

});

// Save the new quiz to MongoDB

const savedQuiz = await newQuiz.save();

console.log('Quiz created successfully:', savedQuiz);

return {

statusCode: 201, // Created

headers: {

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

},

body: JSON.stringify({ message: 'Quiz created successfully', quizTitle: newQuizTitle }),

};

} catch (error) {

console.error('Error creating quiz:', error);

// Check for validation errors

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

return {

statusCode: 400, // Bad Request

headers: {

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

},

body: JSON.stringify({ error: 'Bad Request: ' + error.message }),

};

}

return {

statusCode: 500, // Internal Server Error

headers: {

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

},

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

};

}

};