### LAMBDA FUNCTION -> **paperSubmitCheck**

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

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

Stage -> check

### 1. Role of the API

This API is designed to handle the submission of a quiz paper, marking it as completed if either the MCQ or descriptive sections are fully completed.

### 2. Functioning

* **Token Verification**: Validates the JWT token provided in the request header to ensure authentication.
* **CORS Handling**: Sets appropriate headers to handle Cross-Origin Resource Sharing (CORS).
* **Database Connection**: Connects to MongoDB using Mongoose to interact with the quiz data.
* **Quiz Validation**: Checks if the specified quiz exists and if it is already marked as completed.
* **Completion Check**: Verifies if either the MCQ or descriptive quiz sections are fully completed.
* **Data Storage**: Marks the quiz as completed and saves the updated document in MongoDB.
* **Response Handling**: Returns appropriate success or error responses based on the outcome of the operations.

**3. Request Body**

**{**

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

**"headers": {**

**"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMTc0NjIwLCJleHAiOjE3MjExOTI2MjB9.6NY93YhYC\_5E0gZdnHTiGxNiVU7Lkz1ajbJ5T-KfkvI"**

**}**

**}**

### 4. Response

**Success Response:**

* **Status Code**: 200 {

Paper Submit Successfully

}

**Already Submit: 400**

**{**

**"statusCode": 400,**

**"headers": {**

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

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

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

**},**

**"body": "{\"message\":\"Paper is already submitted\"}"**

**}**

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Unauthorized: token expired"}
* **Status Code**: 404
  + **Body**: {"message": "Quiz not found"}
* **Status Code**: 400
  + **Body**: {"message": "Paper is already submitted"}
  + **Body**: {"message": "Paper is not completed"}
* **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**: Connects to MongoDB using Mongoose for data interaction.
4. **Quiz Validation**: Checks if the quiz exists in the database and if it is already marked as completed.
5. **Completion Check**: Verifies if either the MCQ or descriptive quiz sections are fully completed based on the presence of questions.
6. **Data Storage**: Marks the quiz as completed if the completion check passes and saves it back to MongoDB.
7. **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'); // Assuming you have your Quiz model in a models directory

const MONGO\_URI = process.env.MONGODB\_URI; // Set your MongoDB URI in environment variables

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

const CORS\_HEADERS = {

'Access-Control-Allow-Origin': '\*', // Allow requests from any origin

'Access-Control-Allow-Methods': 'OPTIONS, POST', // Allow OPTIONS and POST methods

'Access-Control-Allow-Headers': 'Content-Type, Authorization', // Allow Content-Type and Authorization headers

};

exports.handler = async (event) => {

try {

const { quizTitle } = JSON.parse(event.body);

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,

headers: CORS\_HEADERS,

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

};

}

const creatorEmail = decoded.email.toLowerCase();

await mongoose.connect(MONGO\_URI);

// Find the quiz

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

if (!quiz) {

return {

statusCode: 404,

headers: CORS\_HEADERS,

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

};

}

// Check if the paper is already submitted

if (quiz.isCompleted) {

return {

statusCode: 400,

headers: CORS\_HEADERS,

body: JSON.stringify({ message: 'Paper is already submitted' }),

};

}

// Check if either mcqQuizz or descriptiveQuizz are fully completed

const isMcqCompleted = quiz.mcqQuizz && quiz.mcqQuizz.length > 0; // Example condition, adjust as needed

const isDescriptiveCompleted = quiz.descriptiveQuizz && quiz.descriptiveQuizz.length > 0; // Example condition, adjust as needed

if (isMcqCompleted || isDescriptiveCompleted) {

quiz.isCompleted = true;

await quiz.save();

return {

statusCode: 200,

headers: CORS\_HEADERS,

body: JSON.stringify({ message: 'Paper submitted successfully' }),

};

} else {

return {

statusCode: 400,

headers: CORS\_HEADERS,

body: JSON.stringify({ message: 'Paper is not completed' }),

};

}

} catch (error) {

console.error(error);

return {

statusCode: 500,

headers: CORS\_HEADERS,

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

};

} finally {

await mongoose.disconnect();

}

};

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;