### LAMBDA FUNCTION -> **quizzdescriptiveQuestionService (excel )**

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

Resource -> **quizzQuestionService ->** descriptiveexcelquestion

Stage -> [**questiontype**](https://us-east-1.console.aws.amazon.com/apigateway/main/apis/ee4pmf8ys1/stages?api=ee4pmf8ys1&region=us-east-1)

### 1. Role of the API

This API is designed to handle the processing and saving of descriptive questions from an Excel file into an existing quiz document stored in MongoDB.

### 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.
* **Excel Data Processing**: Decodes the base64-encoded Excel file data, reads and validates the format, and extracts descriptive questions.
* **Data Validation**: Ensures the quiz title exists and validates the format of the Excel file to match required headers.
* **Descriptive Question Extraction**: Iterates through rows of the Excel file, extracts question and answer pairs, and prepares them for storage.
* **Data Storage**: Appends the new descriptive questions to the existing quiz document in MongoDB and saves the updated document.
* **Response Handling**: Returns appropriate success or error responses based on the outcome of the operations.

**3. Request Body**

{

"headers": {

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

},

"body": {

"excelFile": "base-64-enceded",

"quizTitle": "PrashantTesffft"

}

}

### 4. Response

**Success Response:**

* **Status Code**: 200

{

"statusCode": 200,

"headers": {

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

"Access-Control-Allow-Credentials": true

},

"body": "{\"message\":\"Descriptive questions saved successfully from excel file\",\"totalRows\":4}"

}

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"error": "Token not provided"}
  + **Body**: {"error": "Invalid token"}
* **Status Code**: 400
  + **Body**: {"error": "Quiz title is incorrect"}
  + **Body**: {"error": "Excel file format is incorrect for descriptive questions"}
* **Status Code**: 500
  + **Body**: {"error": "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. **Excel Data Handling**: Decodes and reads the base64-encoded Excel file data, loads it into an ExcelJS workbook, and retrieves the first worksheet.
5. **Header Validation**: Checks if the Excel file has the required headers ('Question' and 'Answer') to ensure the correct format for descriptive questions.
6. **Data Extraction**: Iterates through rows of the Excel worksheet to extract question and answer pairs for descriptive questions.
7. **Quiz Validation**: Checks if the quiz title exists in the database.
8. **Data Storage**: Appends the extracted descriptive questions to the existing quiz document and saves it back to MongoDB.
9. **Response Creation**: Generates and returns appropriate responses based on the success or failure of the operation.

### 6. Dependencies

* **mongoose**: MongoDB object modeling tool.
* **ExcelJS**: Library for reading, manipulating, and writing Excel spreadsheets in JavaScript.
* **jsonwebtoken**: Library for generating and verifying JSON Web Tokens (JWT).

CODE ->

const mongoose = require('mongoose');

const ExcelJS = require('exceljs');

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

const jwt = require('jsonwebtoken');

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

try {

// Verify token

const token = event.headers.Authorization;

if (!token) {

return {

statusCode: 401,

body: JSON.stringify({ error: 'Token not provided' })

};

}

let decoded;

try {

decoded = jwt.verify(token, process.env.JWT\_SECREAT);

// Token is valid, continue with the function

} catch (err) {

return {

statusCode: 401,

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

};

}

// Enable CORS

const headers = {

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

'Access-Control-Allow-Credentials': true

};

// Connect to MongoDB

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

// Decode base64-encoded Excel file data

const excelFileData = Buffer.from(event.body.excelFile, 'base64').toString('binary');

// Read Excel data

const workbook = new ExcelJS.Workbook();

await workbook.xlsx.load(excelFileData);

const worksheet = workbook.worksheets[0]; // Assuming data is in the first worksheet

// Extract quiz title from event

const quizTitle = event.body.quizTitle;

const creatorEmail = decoded.email.toLowerCase();

console.log('this is use mail', creatorEmail);

// Check if the quiz title exists

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

if (!existingQuiz) {

return {

statusCode: 400,

headers,

body: JSON.stringify({ error: 'Quiz title is incorrect' })

};

}

// Validate the format of the Excel file

const headerRow = worksheet.getRow(1);

const requiredHeaders = ['Question', 'Answer'];

const actualHeaders = [headerRow.getCell(1).value, headerRow.getCell(2).value];

const isFormatValid = requiredHeaders.every((header, index) => header === actualHeaders[index]);

if (!isFormatValid) {

return {

statusCode: 400,

headers,

body: JSON.stringify({ error: 'Excel file format is incorrect for descriptive questions' })

};

}

// Iterate through rows and save Descriptive questions to MongoDB

let descriptiveQuizz = [];

for (let rowNumber = 2; rowNumber <= worksheet.rowCount; rowNumber++) {

const row = worksheet.getRow(rowNumber);

// Extract question text and answer

const question = row.getCell(1).value;

const answer = row.getCell(2).value;

// Create Descriptive question data object

const descriptiveData = {

question,

answer

};

// Add Descriptive data to array

descriptiveQuizz.push(descriptiveData);

}

// // Check if there are any valid descriptive questions to save

// if (descriptiveQuizz.length === 0) {

// return {

// statusCode: 400,

// headers,

// body: JSON.stringify({ error: 'No valid descriptive questions found in the Excel file' })

// };

// }

// Append new Descriptive questions to existing quiz document and save it to MongoDB

existingQuiz.descriptiveQuizz.push(...descriptiveQuizz);

await existingQuiz.save();

// Disconnect from MongoDB

await mongoose.disconnect();

// Calculate total number of rows entered through Excel sheet

const totalRows = worksheet.rowCount - 1; // Subtract 1 for header row

return {

statusCode: 200,

headers,

body: JSON.stringify({ message: 'Descriptive questions saved successfully from excel file', totalRows })

};

} catch (error) {

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

return {

statusCode: 500,

headers: {

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

'Access-Control-Allow-Credentials': true

},

body: JSON.stringify({ error: 'Internal server error' })

};

}

};

MODEL -> question.js

const mongoose = require('mongoose');

// Define schema for multiple choice questions

const MCQSchema = new mongoose.Schema({

question: {

type: String,

required: true,

},

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: true,

},

answer: {

type: String,

required: true

},

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();

});

// // Post-save middleware to copy quiz to quiztrash collection

// QuizSchema.post('save', async function(doc, next) {

// try {

// // Use the same schema structure to save the document in a different collection

// const QuizTrash = mongoose.connection.collection('quiztrash');

// await QuizTrash.insertOne(doc.toObject());

// next();

// } catch (error) {

// next(error);

// }

// });

// Create a model using the schema

const Quiz = mongoose.model('Quiz', QuizSchema);

module.exports = Quiz;