### LAMBDA FUCTION NAME -> **quizzQuestionServiceExcel1**

URL -> <https://ee4pmf8ys1.execute-api.us-east-1.amazonaws.com/Many/addExcelQuestion>

Resource -> **quizzQuestionService ->** addExcelQuestion

Stage -> Many

### 1. Role of the API

The primary role of this API is to process an Excel file containing MCQ questions, extract the data, validate it, and append it to 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**: Enables Cross-Origin Resource Sharing (CORS) for the API.
* **Database Connection**: Connects to MongoDB using Mongoose.
* **Excel Data Processing**: Decodes and reads the base64-encoded Excel file, extracts MCQ questions and options, validates the data, and appends the new MCQs 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"

},

"body": {

"excelFile": "base-64-encoded-file",

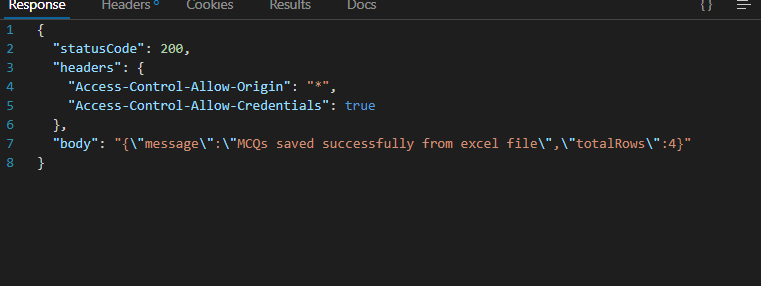
"quizTitle": "PrashantTesffft"

}

}

### 4. Response

**Success Response:**



**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": "Invalid correct answer index in row <row\_number>"}
  + **Body**: {"error": "Incorrect format in row <row\_number>"}
* **Status Code**: 500
  + **Body**: {"message": "<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.
3. **Database Connection**: Establishes a connection to MongoDB.
4. **Excel File Decoding**: Decodes the base64-encoded Excel file data and reads the first worksheet.
5. **Quiz Retrieval**: Searches for an existing quiz document with the provided quizTitle and creatorEmail.
6. **MCQ Processing**: Iterates through the rows of the worksheet, extracts and validates the MCQ data (question, options, correct answer), assigns unique IDs to options, and creates MCQ objects.
7. **Data Validation**: Checks for the presence of at least two options, a valid correct answer index, and ensures the correct format of the data.
8. **Appending MCQs**: Appends the new MCQ objects to the mcqQuizz array of the existing quiz document and saves it to MongoDB.
9. **Response Creation**: Creates and returns appropriate success or error responses based on the outcome of the operations.

### 6. Dependencies

* **mongoose**: For MongoDB object modeling.
* **exceljs**: For reading and processing Excel files.
* **jsonwebtoken**: For JWT token validation.
* **question.js**: Quiz schema model.

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);

} 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();

// 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' })

};

}

// Iterate through rows and save MCQs to MongoDB

let mcqQuizz = [];

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

const row = worksheet.getRow(rowNumber);

// Extract question text

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

// Extract options dynamically

const options = [];

const lastColumn = row.cellCount;

// Determine if the last column is description

let hasDescription = false;

if (typeof row.getCell(lastColumn).value === 'string') {

hasDescription = true;

}

for (let i = 2; i <= (hasDescription ? lastColumn - 2 : lastColumn - 1); i++) {

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

if (answer) {

options.push({ answer, \_id: new mongoose.Types.ObjectId() });

}

}

// Extract correct answer index

const correctAnswerIndex = row.getCell(lastColumn - (hasDescription ? 1 : 0)).value;

// Validate correct answer index

if (typeof correctAnswerIndex !== 'number' || correctAnswerIndex < 1 || correctAnswerIndex > options.length) {

return {

statusCode: 400,

headers,

body: JSON.stringify({ error: `Invalid correct answer index in row ${rowNumber}` })

};

}

// Map correct answer index to option ID

const correctAnswerId = options[correctAnswerIndex - 1].\_id;

// Extract description if present

const description = hasDescription ? row.getCell(lastColumn).value : null;

if (!question || options.length < 2 || !correctAnswerIndex) {

return {

statusCode: 400,

headers,

body: JSON.stringify({ error: `Incorrect format in row ${rowNumber}` })

};

}

// Create MCQ data object

const mcqData = {

question,

options,

correctAnswer: correctAnswerId

};

// Add description to MCQ data if present

if (description) {

mcqData.description = description;

}

// Add MCQ data to array

mcqQuizz.push(mcqData);

}

// Append new MCQs to existing quiz document and save it to MongoDB

existingQuiz.mcqQuizz.push(...mcqQuizz);

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: 'MCQs 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({ message: error.message })

};

}

};

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;