### Lambda Function -> **dashquestionEdit**

API URL ->

<https://7efwp1v3ed.execute-api.us-east-1.amazonaws.com/userdashbordedit/dashbordquestionEdit>

Resource -> **DashbordAPI's ->** [**dashbordquestionEdit**](https://us-east-1.console.aws.amazon.com/apigateway/main/apis/7efwp1v3ed/resources?api=7efwp1v3ed&region=us-east-1)

Stage -> userdashbordedit

### 1. Role of the API

This API updates a quiz question (either MCQ or descriptive) in the database based on the provided questionId.

### 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.
* **Request Validation**: Ensures required fields like questionId and updatedQuestion are present and valid.
* **Question Update**: Updates the quiz question (either MCQ or descriptive) based on the provided questionId.
* **Response Handling**: Returns success or error messages based on the operation's outcome.

**3. Request Body**

**{**

**"headers": {**

**"Authorization": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI2NjcxYWU1NTZhNWY0YTRjNWNhMzMzYjUiLCJlbWFpbCI6InNhaW5pcHM5NDE0NjZAZ21haWwuY29tIiwiaWF0IjoxNzIxMTk3OTA4LCJleHAiOjE3MjEyMTU5MDh9.WxVBGrx8W\_BY2UCaVGwNpth3qux6qA81V\_e-bA2PNKQ"**

**},**

**"body": "{\"questionId\":\"66967ffdfd703b1a7971be2d\",\"updatedQuestion\":{\"question\":\"what is capital of india?\",\"options\":[{\"answer\":\"Jaipur\",\"answerImageLink\":\"https://exambuilder-alfabeto.s3.amazonaws.com/default/1720263495799\_image.jpg\"},{\"answer\":\"Delhi\",\"answerImageLink\":\"\"}],\"correctAnswer\":2,\"description\":\"\",\"questionImageLink\":\"https://exambuilder-alfabeto.s3.amazonaws.com/question/1720263493459\_image.jpg\"}}"**

**}**

### 4. Response

**Success Response:**

* **Status Code**: 200

**{**

**"statusCode": 200,**

**"headers": {**

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

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

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

**},**

**"body": "{\"message\":\"MCQ question updated successfully\",\"questionType\":\"MCQ\"}"**

**}**

**Error Responses:**

* **Status Code**: 401
  + **Body**: {"message": "Unauthorized"} | {"message": "Unauthorized: Invalid token"}
* **Status Code**: 400
  + **Body**: {"message": "Invalid payload structure"} | {"message": "Invalid fields for the question update"}
* **Status Code**: 404
  + **Body**: {"message": "Question not found"}
* **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**: Ensures a consistent connection to MongoDB using Mongoose.
4. **Request Validation**: Ensures required fields like questionId and updatedQuestion are present and valid.
5. **Question Update**: Updates the quiz question (either MCQ or descriptive) based on the provided questionId.
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 jwt = require('jsonwebtoken');**

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

**const DB\_URL = process.env.MONGODB\_URI;**

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

**mongoose.connect(DB\_URL);**

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

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

**if (!token) {**

**return {**

**statusCode: 401,**

**headers: {**

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

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

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

**},**

**body: JSON.stringify({ message: 'Unauthorized' }),**

**};**

**}**

**try {**

**const { questionId, updatedQuestion } = JSON.parse(event.body);**

**let decoded;**

**try {**

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

**} catch (error) {**

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

**return {**

**statusCode: 401,**

**headers: {**

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

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

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

**},**

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

**};**

**}**

**// Validate the structure of the payload**

**if (!questionId || !updatedQuestion || typeof updatedQuestion !== 'object') {**

**return {**

**statusCode: 400,**

**headers: {**

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

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

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

**},**

**body: JSON.stringify({ message: 'Invalid payload structure' }),**

**};**

**}**

**// Ensure either question or questionImageLink is present**

**const hasValidQuestion = updatedQuestion.question || updatedQuestion.questionImageLink;**

**// Determine if the payload is for an MCQ or a descriptive question**

**const isMCQ = (**

**hasValidQuestion &&**

**updatedQuestion.options &&**

**updatedQuestion.correctAnswer &&**

**updatedQuestion.description !== undefined &&**

**Array.isArray(updatedQuestion.options) &&**

**updatedQuestion.options.length >= 2**

**);**

**const isDescriptive = (**

**hasValidQuestion &&**

**updatedQuestion.answer &&**

**!updatedQuestion.options && // Ensure no MCQ fields are present**

**!updatedQuestion.correctAnswer &&**

**!updatedQuestion.description**

**);**

**if (!isMCQ && !isDescriptive) {**

**return {**

**statusCode: 400,**

**headers: {**

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

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

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

**},**

**body: JSON.stringify({ message: 'Invalid fields for the question update' }),**

**};**

**}**

**// Attempt to update the MCQ question**

**let quiz = null;**

**let message = '';**

**if (isMCQ) {**

**// Assign \_id to each option if not already present**

**updatedQuestion.options.forEach(option => {**

**if (!option.\_id) {**

**option.\_id = new mongoose.Types.ObjectId();**

**}**

**});**

**// Convert 1-based index to 0-based and assign option ID to correctAnswer**

**const correctAnswerIndex = updatedQuestion.correctAnswer - 1;**

**updatedQuestion.correctAnswer = updatedQuestion.options[correctAnswerIndex].\_id;**

**quiz = await Quiz.findOneAndUpdate(**

**{ 'mcqQuizz.\_id': questionId },**

**{**

**$set: {**

**'mcqQuizz.$.question': updatedQuestion.question,**

**'mcqQuizz.$.questionImageLink': updatedQuestion.questionImageLink,**

**'mcqQuizz.$.options': updatedQuestion.options,**

**'mcqQuizz.$.correctAnswer': updatedQuestion.correctAnswer,**

**'mcqQuizz.$.description': updatedQuestion.description**

**},**

**$inc: { 'mcqQuizz.$.version': 1 }**

**},**

**{ new: true }**

**);**

**message = 'MCQ question updated successfully';**

**} else if (isDescriptive) {**

**quiz = await Quiz.findOneAndUpdate(**

**{ 'descriptiveQuizz.\_id': questionId },**

**{**

**$set: {**

**'descriptiveQuizz.$.question': updatedQuestion.question,**

**'descriptiveQuizz.$.questionImageLink': updatedQuestion.questionImageLink,**

**'descriptiveQuizz.$.answer': updatedQuestion.answer,**

**'descriptiveQuizz.$.answerImageLink': updatedQuestion.answerImageLink**

**},**

**$inc: { 'descriptiveQuizz.$.version': 1 }**

**},**

**{ new: true }**

**);**

**message = 'Descriptive question updated successfully';**

**}**

**if (!quiz) {**

**return {**

**statusCode: 404,**

**headers: {**

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

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

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

**},**

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

**};**

**}**

**return {**

**statusCode: 200,**

**headers: {**

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

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

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

**},**

**body: JSON.stringify({ message, questionType: isMCQ ? 'MCQ' : 'Descriptive' }),**

**};**

**} catch (error) {**

**console.error(error);**

**return {**

**statusCode: 500,**

**headers: {**

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

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

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

**},**

**body: JSON.stringify({ message: '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: 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**

**}, { 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;**