### LAMBDA FUNCTION -> **studentFormDetails**

API URL -> <https://ybkfar4y6i.execute-api.us-east-1.amazonaws.com/forms/studentFormDetails>

Resource -> **StudentAPI ->** forms

**Role of the API**:

* This API endpoint is responsible for creating a new student record in a MongoDB database.

**Functioning**:

* Upon receiving a POST request with student details, it validates the JWT token in the request headers for authorization.
* Connects to MongoDB using the MONGODB\_URI environment variable.
* Validates the request payload to ensure it contains required fields (fullName, applicationNumber, DOB, AllquestionId).
* Creates a new Student document with the provided details and saves it to the database.
* Returns a success response if the student creation is successful, or an error response if any validation or database operation fails.

 **Request Body**:

* The API expects a JSON object in the request body with the following fields:
  + fullName: Full name of the student.
  + applicationNumber: Application number of the student.
  + DOB: Date of birth of the student.
  + AllquestionId: Array of question IDs associated with the student.

**Response**:

* **201 Created**: If the student is successfully created.
* **401 Unauthorized**: If the JWT token is missing or invalid.
* **500 Internal Server Error**: If there's any server-side issue during student creation.

**Logic**:

* Handles CORS preflight requests (OPTIONS method) to allow CORS headers.
* Validates the JWT token for authorization.
* Connects to MongoDB using Mongoose for database operations.
* Validates the incoming payload to ensure all required fields are present.
* Creates a new Student document and saves it to MongoDB.
* Returns appropriate status codes and messages based on the success or failure of these operations.

**Dependencies**:

* mongoose: MongoDB object modeling tool for Node.js.
* jsonwebtoken: For generating and verifying JSON Web Tokens (JWT).
* Student: Mongoose model representing the Student collection in MongoDB.

CODE ->

const mongoose = require('mongoose');

const jwt = require('jsonwebtoken');

const Student = require('./Student');

// MongoDB Atlas connection management

let conn = null;

const connectToDatabase = async () => {

if (conn == null) {

try {

conn = await mongoose.connect(process.env.MONGODB\_URI)

} catch (error) {

console.error('Error connecting to MongoDB:', error);

throw new Error('Could not connect to MongoDB');

}

}

return conn;

};

// Validate request payload

const validatePayload = (payload) => {

const { fullName, applicationNumber, DOB, AllquestionId } = payload;

if (!fullName || !applicationNumber || !DOB || !AllquestionId) {

throw new Error('All fields (fullName, applicationNumber, DOB, AllquestionId) are required');

}

};

exports.handler = async (event) => {

try {

// Handle preflight CORS request

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

return {

statusCode: 204,

headers: {

'Access-Control-Allow-Origin': '\*', // Allow all origins, update as necessary

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

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

},

body: null,

};

}

// Validate JWT token

const token = event.headers.Authorization;

if (!token) {

return {

statusCode: 401,

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

};

}

let decoded;

try {

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

} catch (err) {

return {

statusCode: 401,

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

};

}

// Connect to MongoDB

await connectToDatabase();

// Parse and validate the request body

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

validatePayload(requestBody);

// Create a new student

const { fullName, applicationNumber, DOB, AllquestionId } = requestBody;

const student = new Student({

fullName,

applicationNumber,

DOB,

AllquestionId,

});

// Save the student to the database

await student.save();

return {

statusCode: 201,

body: JSON.stringify({ message: 'Student created successfully' }),

headers: {

'Access-Control-Allow-Origin': '\*', // Allow all origins

},

};

} catch (error) {

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

return {

statusCode: 500,

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

headers: {

'Access-Control-Allow-Origin': '\*', // Allow all origins

},

};

}

};

MODEL -> Student.js ->

const mongoose = require('mongoose');

const studentSchema = new mongoose.Schema({

fullName: {

type: String,

required: true,

trim: true

},

applicationNumber: {

type: String,

required: true,

unique: true,

trim: true

},

DOB: {

type: Date,

required: true

},

AllquestionId: {

type: String,

required: true,

trim: true

},

generatedShuffledQuestion: {

type: mongoose.Schema.Types.Mixed, // Using Mixed type to allow flexibility in the structure

required: false

},

MCQ: {

AttemptedQuestion: [{

type: mongoose.Schema.Types.Mixed,

required: false

}],

NotAttemptedQuestion: [{

type: mongoose.Schema.Types.Mixed,

required: false

}],

CorrectQuestion: [{

type: mongoose.Schema.Types.Mixed,

required: false

}]

},

descriptiveQuestions: {

AttemptedQuestion: [{

type: mongoose.Schema.Types.Mixed,

required: false

}],

NotAttemptedQuestion: [{

type: mongoose.Schema.Types.Mixed,

required: false

}],

}

},

{

timestamps: true

});

// Setting applicationNumber as the primary key

studentSchema.index({ applicationNumber: 1 }, { unique: true });

const Student = mongoose.model('StudentDetails', studentSchema);

module.exports = Student;