**Back-end Quiz Game API Documentation**

**Overview**

This API allows users to manage and interact with a collection of questions. The available operations include retrieving questions, checking answers, and adding new questions.**1. Get Questions**

**Endpoints**

**1. Get Questions**

Retrieve a list of questions, optionally filtered by difficulty.

GET /getQuestions

**Query Parameters:**

difficulty (string, optional): Filter questions by difficulty level (e.g., easy, medium, hard).

**Response:**

200 OK: An array of questions.

500 Internal Server Error: Error details if there was an issue retrieving the questions.

Example Request:

GET /getQuestions?difficulty=easy

**Example Response:**

[

{

"id": 1,

"question": "What is the capital of France?",

"options": ["Paris", "London", "Rome", "Berlin"],

"correct\_answer": "Paris",

"difficulty": "easy"

},

...

]

**2. Check Answer**

Check if a given answer to a question is correct.

POST /checkAnswer

**Request Body:**

questionId (number, required): The ID of the question.

userAnswer (string, required): The user's answer to the question.

**Response:**

200 OK: Result indicating whether the answer is correct.

404 Not Found: If the question with the given ID is not found.

500 Internal Server Error: Error details if there was an issue processing the answer.

**Example Request:**

POST /checkAnswer

{

"questionId": 1,

"userAnswer": "Paris"

}

**Example Response:**

{

"questionId": 1,

"isCorrect": true

}

**3. Add Question**

Add a new question to the collection.

POST /addQuestion

**Request Body:**

question (string, required): The text of the question.

options (string, required): JSON string of answer options .

correct\_answer (string, required): The correct answer to the question.

difficulty (string, required): The difficulty level of the question (e.g., easy, medium, hard).

**Response:**

201 Created: Confirmation that the question was added successfully.

400 Bad Request: If any required fields are missing.

500 Internal Server Error: Error details if there was an issue adding the question.

**Example Request:**

POST /addQuestion

{

"question": "What is 2 + 2?",

"options": [“a":”2”,”b”: "3", “c”:"4", “d":”5"],

"correct\_answer": "c",

"difficulty": "easy"

}

**Example Response:**

{

"success": true,

"message": "Question added successfully"

}

Error Handling

All endpoints will return a JSON object with an error field containing the error message in case of any issues.

**Example Error Response:**

{

"error": "An error occurred",

"message": "Missing required fields"

}

Routing

router.get('/getQuestions', getQuestions);

router.post('/checkAnswer', checkAnswer);

router.post('/addQuestion', addQuestion);

module.exports = router;

**Helper Functions**

The updateQuestions function is to handle updating the questions JSON file.