

# Quiz Mono Repo API

## 1.1 Add Question

### Endpoint:

POST /question

### Description:

Creates a new question in the system.

### Request Body (JSON):

json

```
{  
  "questionTitle": "What is JVM?",  
  "option1": "Java Virtual Machine",  
  "option2": "Java Very Much",  
  "option3": "Just Virtual Memory",  
  "option4": "Java Virtual Memory",  
  "correctAnswer": "Java Virtual Machine",  
  "difficultyLevel": "EASY",  
  "category": "JAVA"  
}
```

### Response (201 Created):

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/question`
- Method:** `POST`
- Body:** A JSON object with the following structure:

```
1 {
2
3   "questionTitle": "parent class of all classes in java?",
4   "option1": "list",
5   "option2": "object",
6   "option3": "String",
7   "option4": "Integer",
8   "correctAnswer": "object",
9   "difficultyLevel": "EASY",
10  "category": "JAVA"
11 }
```
- Response:** `200 OK` with a status bar showing `335 ms` and `370 B`. The response body is a JSON object:

```
1 {
2   "id": 2,
3   "questionTitle": "parent class of all classes in java?",
4   "option1": "list",
5   "option2": "object",
6   "option3": "String",
7   "option4": "Integer",
8   "correctAnswer": "object",
9   "difficultyLevel": "EASY",
10  "category": "JAVA"
11 }
```

## 1.2 Get All Questions (Paginated)

### Endpoint:

GET `/question?page={page}&size={size}`

### Description:

Retrieves all questions with pagination.

### Example Request:

GET `/question?page=0&size=5`

### Response:

http://localhost:8080/question?page=1&size=10

GET http://localhost:8080/question?page=1&size=10

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
page	1	
size	10	

Body Cookies Headers (5) Test Results

200 OK • 370 ms • 3.15 KB

```
{
  "content": [
    {
      "id": 11,
      "questionTitle": "Which keyword is used to inherit a class in Java?",
      "option1": "implements",
      "option2": "extends",
      "option3": "inherits",
      "option4": "super",
      "correctAnswer": "extends",
      "difficultyLevel": "EASY",
      "category": "JAVA"
    }
  ]
}
```

Postbot Runner Start Proxy Cookies Vault Trash

### 1.3 Filter Questions by Category & Difficulty

#### Endpoint:

GET /question/filter?category={category}&difficulty={difficulty}

#### Example Request:

GET /question/filter?category=JAVA&difficulty=EASY

#### Response:

HTTP <http://localhost:8080/question/filter?category=JAVA&difficulty=EASY> Save Share

GET <http://localhost:8080/question/filter?category=JAVA&difficulty=EASY> Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> category	JAVA			
<input checked="" type="checkbox"/> difficulty	EASY			
	Key	Value		Description

Body Cookies Headers (5) Test Results 200 OK • 117 ms • 2.21 KB

{ } JSON Preview Visualize

```
1 [
2   {
3     "id": 1,
4     "questionTitle": "What does JVM stand for?",
5     "option1": "Java Virtual Machine",
6     "option2": "Java Variable Method",
7     "option3": "Just Virtual Memory",
8     "option4": "Joint Variable Manager",
9     "correctAnswer": "Java Virtual Machine",
10    "difficultyLevel": "EASY",
11    "category": "JAVA"
12  },
13 ]
```

## 1.4 Update Question Partially

### Endpoint:

PATCH /question/{id}

### Description:

Partially updates question details.

### Request Body Example:

#### Json:

```
{
  "questionTitle": "Explain JVM in Java",
  "difficultyLevel": "MEDIUM"
}
```

### Response:

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/question/1`
- Method:** PATCH
- Body:**

```
{
  "difficultyLevel": "MEDIUM"
}
```
- Response:** 200 OK, 200 ms, 432 B. The response body is a JSON object: 

```
{
  "id": 1,
  "questionTitle": "What does JVM stand for?",
  "option1": "Java Virtual Machine",
  "option2": "Java Variable Method",
  "option3": "Just Virtual Memory",
  "option4": "Joint Variable Manager",
  "correctAnswer": "Java Virtual Machine",
  "difficultyLevel": "MEDIUM",
  "category": "JAVA"
}
```

## 1.5 Delete Question

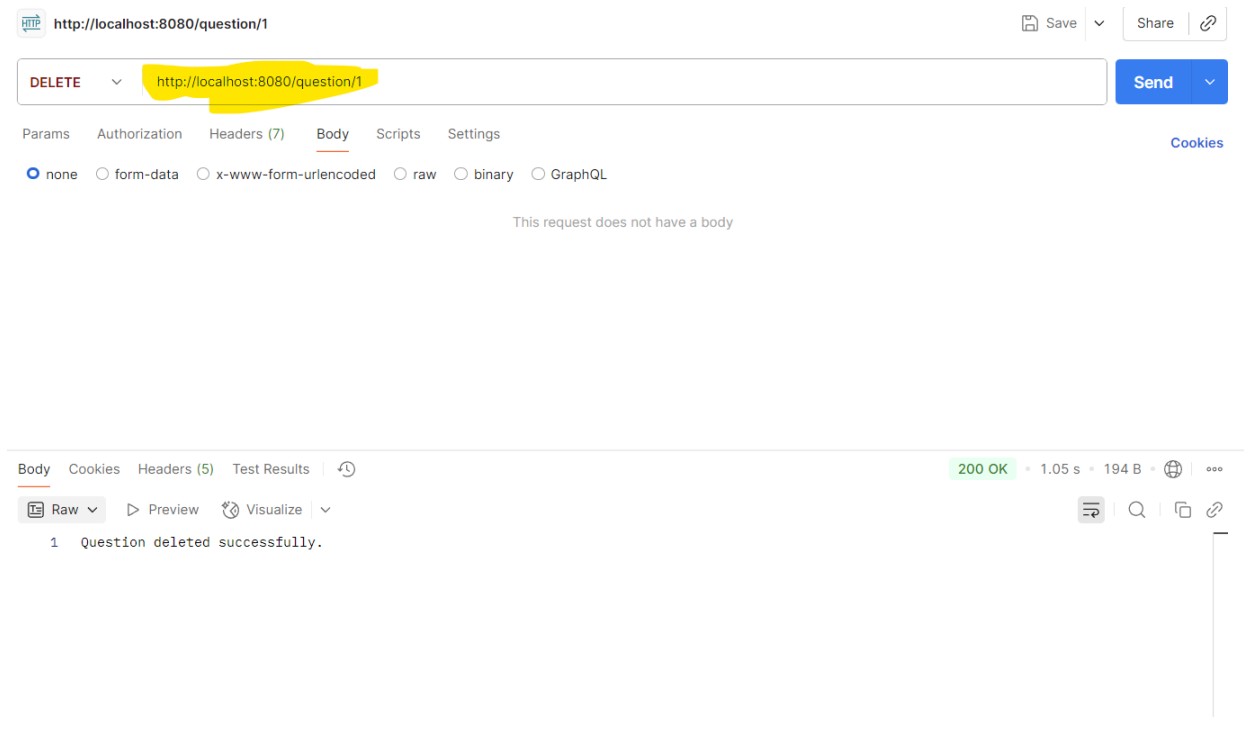
### Endpoint:

DELETE /question/{id}

### Example:

DELETE /question/1

### Response:



# Quiz APIs:

## 2.1 Create Quiz

### Endpoint:

POST /quiz

### Request Body:

#### Json

```
{  
  "quizTitle": "Java Basics Quiz",  
  "questionIds": [1, 2, 3]  
}
```

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/quiz`
- Method:** `POST`
- Body:**

```
{
  "quizTitle": "JavaCompetition",
  "questionIds": [2,3,4,5]
}
```
- Response:** `200 OK` (154 ms, 1.06 KB)
- Response Body (JSON):**

```
{
  "id": 1,
  "quizTitle": "JavaCompetition",
  "questions": [
    {
      "id": 2,
      "questionTitle": "parent class of all classes in java?",
      "option1": "list",
      "option2": "object",
      "option3": "String",
      "option4": "Integer",
      "correctAnswer": "object",
    }
  ]
}
```

## 2.2 Get Quiz by ID

### Endpoint:

`GET /quiz/{id}`

### Example:

`GET /quiz/1`

### Response:

The screenshot shows a web browser interface for a REST client. The URL bar displays `http://localhost:8080/quiz/1`. The request method is set to `GET`. The response status is `200 OK` with a response time of `64 ms` and a size of `1.06 KB`. The response body is displayed in JSON format:

```
{
  "id": 1,
  "quizTitle": "JavaCompetition",
  "questions": [
    {
      "id": 2,
      "questionTitle": "parent class of all classes in java?",
      "option1": "list",
      "option2": "object",
      "option3": "String",
      "option4": "Integer",
      "correctAnswer": "object",
    }
  ]
}
```

## 2.3 Submit Quiz

### Endpoint:

POST `/quiz/{id}/submit`

### Request Body Example:

#### Json:

```
{
  "answers": {
    "1": "Java Virtual Machine",
    "2": "Spring Framework"
  }
}
```

### Response:



The screenshot shows a web browser interface for testing HTTP requests. The URL bar displays `http://localhost:8080/quiz/1/submit`. The request method is set to `POST`, and the request body is a JSON object:

```
1 {
2   "answers": {
3     "2": "object",
4     "3": "boolean",
5     "4": "1",
6     "5": "start()"
7   }
8 }
9 }
```

The response status is `200 OK` with a response time of `12 ms` and a size of `224 B`. The response body is a JSON object:

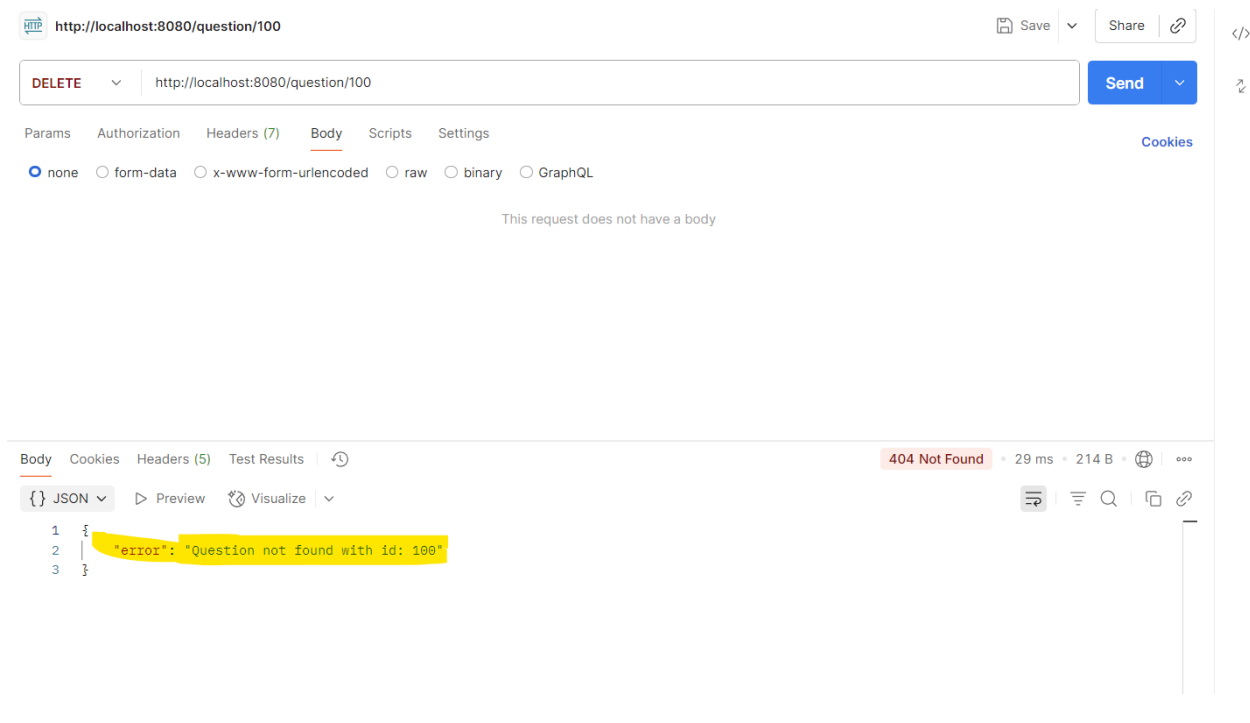
```
1 {
2   "totalQuestions": 4,
3   "correctAnswers": 3,
4   "scorePercentage": 75
5 }
```

## Extra Features:

### 1) Custom Exception Handling

1. Implemented **@ControllerAdvice** to centralize exception handling for all controllers.
2. Created **custom exceptions** such as `ResourceNotFoundException` and `InvalidRequestException` to return meaningful error messages.
3. All errors are returned in **consistent JSON format** with message, **proper HTTP status codes** (404, 400, etc.).

### Example Error Response:



## 2) Aspect-Oriented Programming (AOP)

1. Added **logging aspect** using `@Aspect` and `@Around` to log API execution time, method entry/exit, and parameters.
2. Helps in **monitoring performance** and debugging without modifying controller/service logic.
3. Can be extended for **security checks, transaction handling, or auditing** in the future.

### Example Log Output:



## 3) Swagger API Documentation

1. Integrated **Springdoc OpenAPI (Swagger UI)** to generate interactive API documentation automatically.

2. Available at /swagger-ui.html or /swagger-ui/index.html for easy testing of endpoints.
3. Includes **request/response schemas**, **enum values**, and example bodies for each API.

The image displays two screenshots of the Swagger UI for the Quiz Mono Repo API, version 1.0.0, OAS 3.0. The top screenshot shows the 'questions' definition, and the bottom screenshot shows the 'quiz' definition. Both screenshots include a 'Servers' section with the URL 'http://localhost:8080 - Generated server url' and a list of endpoints for the respective controllers.

**Top Screenshot: questions definition**

API documentation for Quiz and Question Management System

Servers: <http://localhost:8080> - Generated server url

**question-controller**

- GET /question
- POST /question
- DELETE /question/{id}
- PATCH /question/{id}
- GET /question/filter

**Bottom Screenshot: quiz definition**

API documentation for Quiz and Question Management System

Servers: <http://localhost:8080> - Generated server url

**quiz-controller**

- POST /quiz
- POST /quiz/{id}/submit
- GET /quiz/{id}