12.Quiz App Submit

1. QuizController: Submit Quiz Logic

```
@RestController
@RequestMapping("quiz")
public class QuizController {

    @Autowired
    QuizService quizService;

    // Endpoint to create a quiz using query parameters
    @PostMapping("create")
    public ResponseEntity<String> createQuiz(
            @RequestParam String category,
            @RequestParam int numQ,
            @RequestParam String title) {
        return quizService.createQuiz(category, numQ, title);
    }

    // Endpoint to get quiz questions by quiz ID
    @GetMapping("/get/{id}")
    public ResponseEntity<List<QuestionWrapper>> getQuizQuestions(@PathVariable Integer id) {
        List<QuestionWrapper> questionsForUser = quizService.getQuizQuestions(id);
        return new ResponseEntity<>(questionsForUser, HttpStatus.OK);
    }

    // Endpoint to submit quiz answers and calculate the result
    @PostMapping("submit/{id}")
    public ResponseEntity<Integer> submitQuiz(@PathVariable Integer id, @RequestBody
List<Response> response) {
        return quizService.calculateResult(id, response);
    }
}
```

- **submitQuiz Method**: This method takes the quiz ID and a list of user responses (in the form of Response objects) and calls the calculateResult method from QuizService to evaluate the score. The result (number of correct answers) is returned as an integer with HttpStatus.OK.

2. QuizService: Result Calculation Logic

```
@Service
public class QuizService {
  @Autowired
  QuizRepository quizRepository;
  @Autowired
  QuestionRepository questionRepository;
  // Create a new quiz using the specified category, number of questions, and title
  public ResponseEntity<String> createQuiz(String category, int numQ, String title) {
     try {
        // Fetch random questions by category
        List<Question> questions = questionRepository.findRandomQuestionsByCategory(category,
numQ);

        // Check if enough questions are retrieved
        if (questions.isEmpty()) {
           return new ResponseEntity<>("Not enough questions available for the given category",
HttpStatus.BAD_REQUEST);
        }
        // Create a new Quiz object
        Quiz quiz = new Quiz();
        quiz.setTitle(title);
        quiz.setQuestions(questions);
        // Save the quiz
        quizRepository.save(quiz);
        return new ResponseEntity<>("Quiz created successfully with title: " + title,
HttpStatus.CREATED);
     } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>("Error occurred while creating the quiz",
HttpStatus.INTERNAL_SERVER_ERROR);
     }
  }
  // Fetch all quizzes
  public ResponseEntity<List<Quiz>> getAllQuizzes() {
     try {
        return new ResponseEntity<>(quizRepository.findAll(), HttpStatus.OK);
     } catch (Exception e) {
        e.printStackTrace();
        return new ResponseEntity<>(new ArrayList<>(), HttpStatus.INTERNAL_SERVER_ERROR);
     }
  }

  // Method to get the questions by quiz ID
  public List<QuestionWrapper> getQuizQuestions(Integer id) {
```

```
    // Fetch the quiz from the database
    Optional<Quiz> quiz = quizRepository.findById(id);
    // Check if the quiz is present
    if (quiz.isPresent()) {
        // Get the questions from the quiz object
        List<Question> questionsFromDB = quiz.get().getQuestions();
        List<QuestionWrapper> questionsForUser = new ArrayList<>();
        // Loop through each question and convert it to QuestionWrapper
        for (Question q : questionsFromDB) {
            QuestionWrapper qw = new QuestionWrapper(
                    q.getId(),
                    q.getQuestionTitle(),
                    q.getOption1(),
                    q.getOption2(),
                    q.getOption3(),
                    q.getOption4()
                );
            questionsForUser.add(qw);  // Add each mapped question to the list
        }
        return questionsForUser;
    } else {
        // Handle case when the quiz is not found (you can throw an exception or return an empty list)
        throw new ResourceNotFoundException("Quiz not found with id: " + id);
    }
  }
  public ResponseEntity<Integer> calculateResult(Integer id, List<Response> responses) {
    Quiz quiz = quizRepository.findById(id).get();
    List<Question> questions = quiz.getQuestions();
    int right = 0;
    int i = 0;
    for (Response response : responses) {
        if (response.getResponse().equals(questions.get(i).getRightAnswer())) {
            right++;
        }
        i++;
    }
    return new ResponseEntity<>(right, HttpStatus.OK);
  }
}
```

- **calculateResult Method**:
    - Fetches the quiz by its ID using quizRepository.
    - Retrieves the list of questions associated with the quiz.
    - Loops through the provided user responses (Response objects) and compares each one with the correct answer from the corresponding question.
    - Keeps a counter (right) to track the number of correct answers.
    - Returns the total number of correct answers as a response.

3. Response Entity

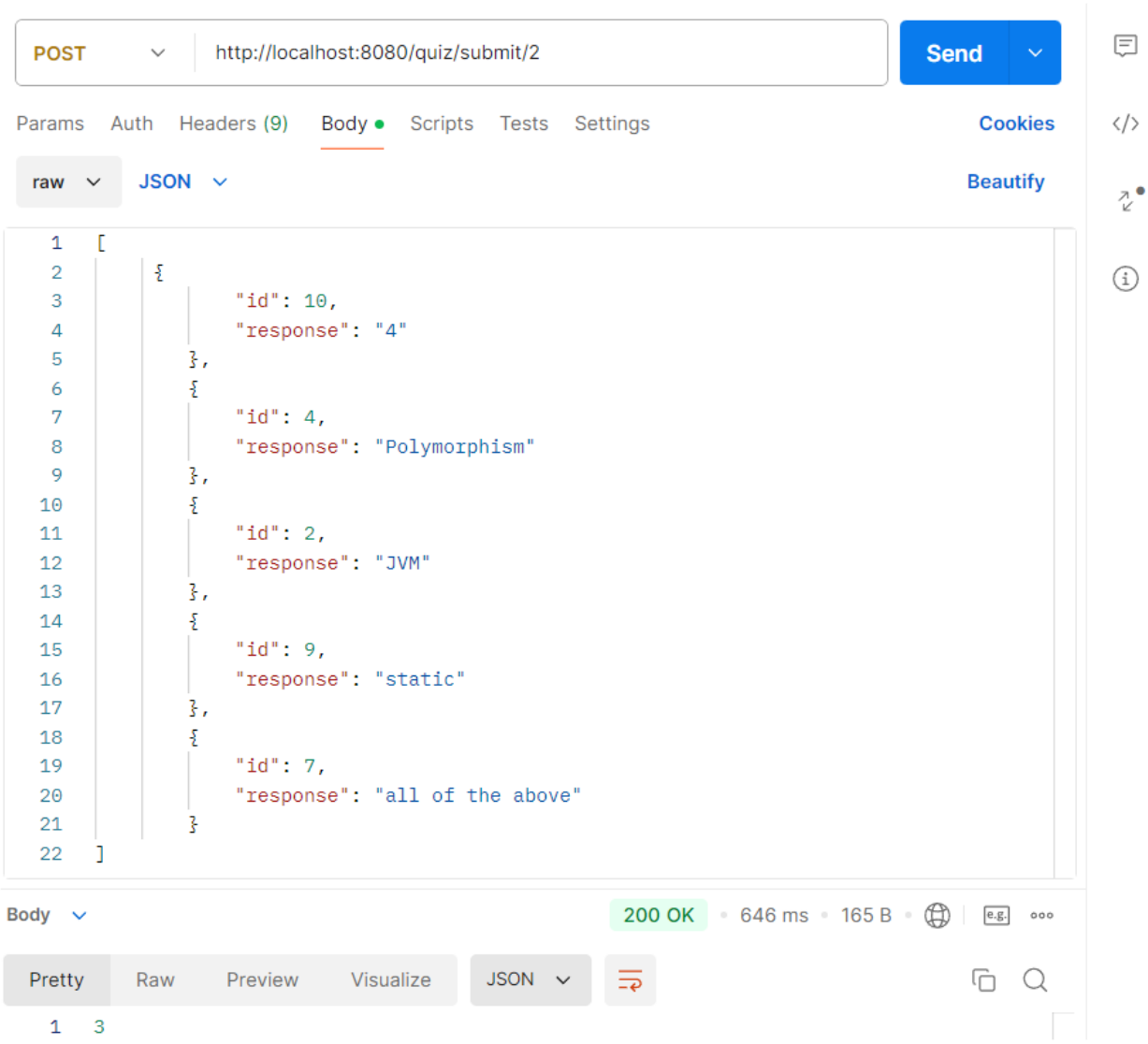
```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Response {
    private int id;         // Question ID
    private String response; // User's response
}
```

- **Response Class**: Represents the user's answer to a question. Each response contains the question's id and the user's response. This is passed as a request body to the submitQuiz endpoint.

Flow Explanation

1. **Submit Quiz**: When the user submits their answers through the submitQuiz endpoint, their responses (as a list of Response objects) are sent to the backend.
2. **Service Logic**: The calculateResult method compares each user's answer with the correct answer stored in the Question entity.
3. **Result Calculation**: A count of correct answers is maintained and returned to the user.

Postman:



For Application:

1. API LINK
2. Quiz App Monolithic Project
3. Readme
4. Link: [Project Link]