11.Quiz part 3

QuestionWrapper Class

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class QuestionWrapper {

    private Integer id;
    private String questionTitle;
    private String option1;
    private String option2;
    private String option3;
    private String option4;
}
```

Explanation:

- **Purpose**: The QuestionWrapper class is a Data Transfer Object (DTO) used to send limited details about a Question entity (excluding sensitive information such as answers) to the client.
- **Fields**:
    - id: The unique identifier for the question.
    - questionTitle: The text or title of the question.
    - option1, option2, option3, option4: The four possible options for the question.
- **Why use a DTO?**
    - This is useful for encapsulating data without exposing certain details (like the correct answer). When sending data to users (e.g., for quizzes), you often don't want to include sensitive information, like the correct answers, in the response.

Updated QuizService with getQuizQuestions Method

```
@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);
        }
    }

    // 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
            throw new ResourceNotFoundException("Quiz not found with id: " + id);
        }
    }
}
```

Explanation:

- **getQuizQuestions(Integer id)**: This method retrieves a list of questions for a specific quiz (by quiz ID) and converts them into QuestionWrapper objects to avoid exposing sensitive details (like correct answers).
    - **Fetch Quiz**: It first fetches the quiz by ID from the QuizRepository.
    - **Convert Questions**: If the quiz exists, it loops through the associated questions, converting each Question entity into a QuestionWrapper by extracting only the necessary fields (id, questionTitle, option1, option2, option3, option4).
    - **Return**: The method then returns a list of QuestionWrapper objects.

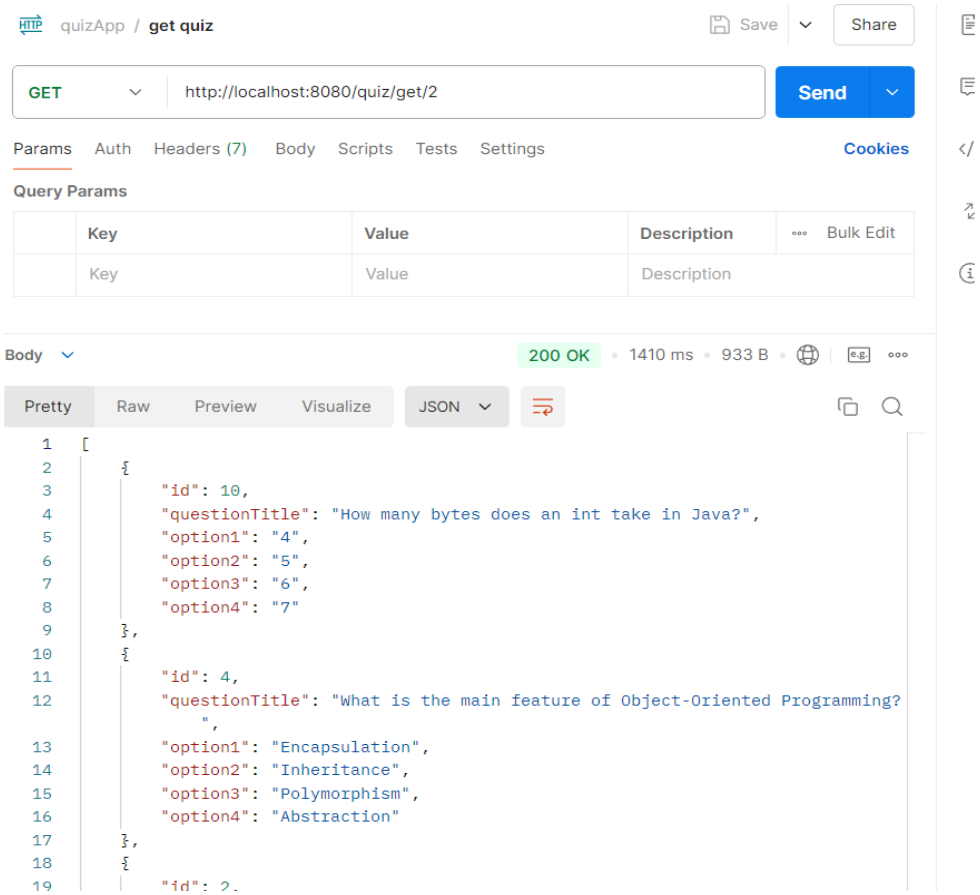
Updated QuizController with getQuizQuestions Endpoint

```
@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);
    }

    @GetMapping("/get/{id}")
    public ResponseEntity<List<QuestionWrapper>> getQuizQuestions(@PathVariable Integer id) {
        // Call the service to get the questions
        List<QuestionWrapper> questionsForUser = quizService.getQuizQuestions(id);

        // Return the response with the list of QuestionWrapper DTOs and HTTP status 200 (OK)
        return new ResponseEntity<>(questionsForUser, HttpStatus.OK);
    }
}
```

Explanation:

- **@GetMapping("/get/{id}")**: This endpoint is mapped to retrieve the quiz questions based on the quiz ID. It internally calls the getQuizQuestions method from QuizService.
    - **Response**: The response will contain a list of QuestionWrapper objects with HTTP status 200 (OK).
    - **Why use QuestionWrapper in the response?**
        - The client only needs limited information to display quiz questions (like the question title and options). By using QuestionWrapper, you can avoid sending irrelevant or sensitive data (e.g., correct answers) to the client.

Postman:



Conclusion

- **Why use QuestionWrapper?**
  - **Separation of concerns**: By using QuestionWrapper, you can keep the business logic of your application (like storing questions and answers) separate from the data you expose to users.
  - **Security**: You avoid sending answers or other sensitive data unnecessarily to users, which is important for quiz applications.
  - **Flexibility**: If the client-side requirements change (e.g., they want to see more or less data), you can easily modify the QuestionWrapper without affecting your core Question entity or database schema.