Endpoint Being Tested: http://127.0.0.1:5000/extra_questions

Case: *Successfully generated extra practice questions*

Request Method: POST

Inputs:

```
{
  "lecture_id": 2
}
```

Expected Output:

```
HTTP Status Code: 200 and JSON with 'lecture_id' and 'questions'
```

Actual Output:


```
HTTP Status Code: 200
JSON: {"lecture_id": 2, "questions": [{"question": "The lecturer uses Replit.com
to demonstrate Python programming.  What crucial advantage does this platform
offer beginners?", "options": {"A": "It requires no prior software installation.",
"B": "It automatically corrects syntax errors.", "C": "It provides pre-built AI
assistance for coding.", "D": "It offers advanced debugging tools unavailable
elsewhere."}, "correct_answer": "A"}, {"question": "In the context of the
lecture's example, what does the 'repository' in Replit represent?", "options":
{"A": "A folder containing system files.", "B": "A temporary storage space for
code execution.", "C": "A location for storing and managing code.", "D": "A pre-
defined set of programming exercises."}, "correct_answer": "C"}, {"question": "The
lecturer demonstrates creating a 'staircase' pattern using asterisks (*). What
fundamental programming concept does this example primarily illustrate?",
"options": {"A": "The use of complex algorithms.", "B": "The importance of error
handling.", "C": "Sequential execution of commands.", "D": "The limitations of
simple print statements."}, "correct_answer": "C"}, {"question": "The lecturer
highlights the difficulty of creating the reverse staircase pattern manually. What
core programming challenge does this exemplify?", "options": {"A": "The complexity
of string manipulation.", "B": "The limitations of the print function.", "C": "The
need for automation and iteration.", "D": "The inherent difficulty of reverse
engineering code."}, "correct_answer": "C"}, {"question": "The concluding
question, 'Is there any way we can automate this?', foreshadows the introduction
of which important programming concept in future lectures?", "options": {"A":
"Object-oriented programming.", "B": "Data structures and algorithms.", "C":
"Loops and iterative processes.", "D": "Advanced debugging techniques."},
"correct_answer": "C"}]}
```

Result: Success

Pytest Code:

```
def test_extra_questions_success(client):
    payload = {
        "lecture_id": 2
    }
    response = client.post("/extra_questions", json=payload)
    data = response.get_json()

    expected_status = 200
    result = "Success" if (
        response.status_code == expected_status
        and "lecture_id" in data
        and "questions" in data
        and isinstance(data["questions"], list)
    ) else "Failed"

    write_test_doc(
        title="***Case:*** *Successfully generated extra practice questions*",
        endpoint="http://127.0.0.1:5000/extra_questions",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 200 and JSON with 'lecture_id' and
'questions'",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 200
    assert "lecture_id" in data
    assert "questions" in data
    assert isinstance(data["questions"], list)
```

Case: *Transcript not found for lecture ID*

Request Method: POST

Inputs:

```
{
  "lecture_id": 99999999
}
```

Expected Output:

```
HTTP Status Code: 404 and error message
```

Actual Output:

```
HTTP Status Code: 404
JSON: {"Error": "Transcript not found or empty"}
```

Result: Success

Pytest Code:

```
def test_extra_questions_not_found(client):
    payload = {
        "lecture_id": 99999999  # Use an ID that returns no transcript
    }
    response = client.post("/extra_questions", json=payload)
    data = response.get_json()

    expected_status = 404
    result = "Success" if response.status_code == expected_status and "Error" in
data else "Failed"

    write_test_doc(
        title="***Case:*** *Transcript not found for lecture ID*",
        endpoint="http://127.0.0.1:5000/extra_questions",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 404 and error message",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 404
    assert data and "Error" in data
```

Case: *Internal server error while generating questions*

Request Method: POST

Inputs:

```
{
  "lecture_id": -999
}
```

Expected Output:

```
HTTP Status Code: 500 and error message
```

Actual Output:

```
HTTP Status Code: 404
JSON: {"Error": "Transcript not found or empty"}
```

Result: Failed

Pytest Code:

```
def test_extra_questions_internal_error(client):
    payload = {
        "lecture_id": -999  # Simulate internal server error using invalid ID
    }
    response = client.post("/extra_questions", json=payload)
    try:
        data = response.get_json()
    except Exception:
        data = None

    expected_status = 500
    result = "Success" if response.status_code == expected_status else "Failed"

    write_test_doc(
        title="***Case:*** *Internal server error while generating questions*",
        endpoint="http://127.0.0.1:5000/extra_questions",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 500 and error message",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 500
    assert data and "error" in data if isinstance(data, dict) else True
```
