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

Case: *Successful lecture review report generation*

Request Method: POST

Inputs:

```
{
  "course_id": 1,
  "week": 1
}
```

Expected Output:

```
HTTP Status Code: 200 and JSON with 'lecture_review_report'
```

Actual Output:


```
HTTP Status Code: 200
JSON: {"course_id": 1, "week": 1, "lecture_review_report": "**Common
Issues:**\n\n* **Clarity and Examples:**  Students consistently mention a lack of
clarity and request more examples to illustrate concepts.  This is evident in the
feedback for Lecture 3 (\"nothing was clear\") and implicitly in the audio issue
impacting understanding in Lecture 2.\n* **Audio Quality:**  Poor audio quality
was specifically mentioned in Lecture 2's feedback.\n* **Pace:**  Lecture 1 was
considered good but too slow. This suggests a need to balance pacing to cater to
different learning styles.\n\n\n**Suggested Improvements:**\n\n* **Incorporate
More Examples:**  The instructor should include more diverse and relevant examples
in lectures to improve understanding and engagement.  Consider using real-world
applications, case studies, and visual aids to illustrate key concepts.\n*
**Improve Audio Quality:**  Check microphone quality and settings before lectures.
Consider using a higher-quality microphone and ensuring proper room acoustics to
minimize background noise and improve audio clarity.  If recording lectures,
invest in audio editing software to improve post-production sound.\n* **Optimize
Lecture Pace:**  Find a balance between a pace that allows for thorough
explanation and one that maintains student engagement. Vary the pace throughout
the lecture, using different teaching methods to cater to diverse learning styles.
Consider incorporating interactive elements like short quizzes or discussions to
break up longer explanations.  Solicit feedback on pacing during lectures.\n*
**Seek Regular Feedback:** Implement a system for collecting and acting upon
student feedback throughout the course, not just at the end. This could involve
brief in-class polls, online feedback forms, or short anonymous surveys after each
lecture.\n\n\n**Overall Student Satisfaction:**\n\nStudent satisfaction appears
somewhat mixed. While two lectures received a 4/5 rating, indicating a generally
```

```
positive experience, one lecture received a significantly lower 2/5 rating,
highlighting a considerable issue with clarity and examples.  This suggests that
while some aspects of the lectures are well-received, significant improvements are
needed to ensure consistent quality and student engagement across all lectures.
The average rating is 3.33/5, indicating room for substantial improvement."}
```

Result: Success

Pytest Code:

```
def test_feedback_success(client):
    payload = {"course_id": 1, "week": 1}
    response = client.post("/lecture_review_report", json=payload)
    try:
        data = response.get_json()
    except Exception:
        data = None

    expected_status = 200
    result = "Success" if response.status_code == expected_status and
"lecture_review_report" in data else "Failed"

    write_test_doc(
        title="***Case:*** *Successful lecture review report generation*",
        endpoint="http://127.0.0.1:5000/lecture_review_report",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 200 and JSON with 'lecture_review_report'",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 200
    assert "lecture_review_report" in data
```

Case: *Missing required fields (course_id)*

Request Method: POST

Inputs:

```
{}
```

Expected Output:

```
HTTP Status Code: 400 and error message
```

Actual Output:

```
HTTP Status Code: 400
JSON: {"message": {"course_id": "Course ID is required"}}
```

Result: Success

Pytest Code:

```
def test_missing_fields(client):
    payload = {}  # Missing course_id
    response = client.post("/lecture_review_report", json=payload)
    try:
        data = response.get_json()
    except Exception:
        data = None

    expected_status = 400
    result = "Success" if response.status_code == expected_status else "Failed"

    write_test_doc(
        title="***Case:*** *Missing required fields (course_id)*",
        endpoint="http://127.0.0.1:5000/lecture_review_report",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 400 and error message",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 400
```

Case: *Internal server error on review report generation*

Request Method: POST

Inputs:

```
{
  "course_id": "INVALID",
  "week": 1
}
```

Expected Output:

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

Actual Output:

```
HTTP Status Code: 400
JSON: {"message": {"course_id": "Course ID is required"}}
```

Result: Failed

Pytest Code:

```python
def test_feedback_internal_error(client):
    # Invalid data to trigger internal server error (e.g., string instead of int
for course_id)
    payload = {"course_id": "INVALID", "week": 1}
    response = client.post("/lecture_review_report", 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 on review report generation*",
        endpoint="http://127.0.0.1:5000/lecture_review_report",
        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
```
