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

Case: *Successful execution and feedback generation*

Request Method: POST

Inputs:

```
{
  "user_id": 1,
  "code": "def greet(name):\n    print(f'Hello, {name}')\n\ngreet('World')"
}
```

Expected Output:

```
HTTP Status Code: 200 and JSON with 'execution_result' and 'feedback'
```

Actual Output:

```
HTTP Status Code: 200
JSON: {"execution_result": "Hello, World", "error": null, "feedback": "- **Errors
Found:**  No apparent syntax, indentation, or runtime errors are present in the
provided code snippet.  However, the code lacks robustness and flexibility which
could lead to issues with different inputs or usage contexts.\n\n\n- **Suggested
Improvements:** The function currently only prints to the console. Consider
exploring ways to make the function more versatile, perhaps allowing it to return
a value instead of solely printing.  The handling of the input `name` could be
improved to gracefully handle unexpected input types or empty strings.\n\n\n-
**Best Practices:**  While functional, the code could benefit from more
descriptive variable names if a more complex application were to use it.  Adding
docstrings to explain the function's purpose and parameters would enhance
readability and maintainability, especially in larger projects.  Consider using
more robust input validation to ensure the function handles various scenarios
gracefully."}
```

Result: Success

Pytest Code:

```
def test_successful_code_feedback(client):
    payload = {
        "user_id": 1,
        "code": "def greet(name):\n    print(f'Hello, {name}')\n\ngreet('World')"
    }
    response = client.post("/assignment_feedback", json=payload)
    data = response.get_json()

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

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

    assert response.status_code == 200
    assert "execution_result" in data
    assert "feedback" in data
```

Case: *Code with syntax error (missing colon)*

Request Method: `POST`

Inputs:

```
{
  "user_id": 2,
  "code": "def add(a, b)\n    return a + b\n\nprint(add(5, 3))"
}
```

Expected Output:

```
HTTP Status Code: 200 and JSON with 'error' and 'feedback'
```

Actual Output:

```
HTTP Status Code: 200
JSON: {"execution_result": "", "error": {"type": "SyntaxError", "message":
"expected ':' (<string>, line 1)"}, "feedback": "- **Errors Found:**\n\nThe code
contains a syntax error related to the function definition and a potential
indentation error.  The `return` statement's placement relative to the function
definition is crucial and needs to be carefully examined.  Additionally, a missing
colon after the function definition's parameter list is a clear syntax error that
prevents execution.\n\n\n- **Suggested Improvements:**\n\nThe function definition
should be reviewed to ensure the correct syntax for defining a function in Python.
Pay close attention to the placement of the `return` statement and the use of
colons to properly delimit code blocks. The use of consistent and appropriate
indentation is vital for Python code readability and execution.  Consider adding
docstrings to clearly explain the function's purpose and parameters.\n\n\n- **Best
Practices:**\n\nUsing a consistent indentation style (e.g., 4 spaces) throughout
the code improves readability.  Adding a docstring to the `add` function would
enhance its understandability and maintainability.  Employing meaningful variable
names would further improve the clarity of the code. Consider adding basic input
validation to handle potential errors or unexpected input types.  Finally,  more
comprehensive testing would help identify potential issues earlier in the
development process."}
```

Result: Success

Pytest Code:

```
def test_code_with_syntax_error(client):
    payload = {
        "user_id": 2,
        "code": "def add(a, b)\n    return a + b\n\nprint(add(5, 3))"
    }

    response = client.post("/assignment_feedback", json=payload)
    data = response.get_json()

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

    write_test_doc(
        title="***Case:*** *Code with syntax error (missing colon)*",
        endpoint="http://127.0.0.1:5000/assignment_feedback",
        method="POST",
        inputs=json.dumps(payload, indent=2),
        expected="HTTP Status Code: 200 and JSON with 'error' and 'feedback'",
        actual=f"HTTP Status Code: {response.status_code}\nJSON:
{json.dumps(data)}",
        result=result
    )

    assert response.status_code == 200
    assert "error" in data
    assert "feedback" in data
```

Case: *Empty code submission*

Request Method: `POST`

Inputs:

```
{
  "user_id": 3,
  "code": ""
}
```

Expected Output:

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

Actual Output:

```
HTTP Status Code: 400
JSON: {"error": "Code cannot be empty"}
```

Result: `Success`

Pytest Code:

```
def test_empty_code_submission(client):
    payload = {
        "user_id": 3,
        "code": ""
    }

    response = client.post("/assignment_feedback", json=payload)
    data = response.get_json()

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

    write_test_doc(
        title="***Case:*** *Empty code submission*",
        endpoint="http://127.0.0.1:5000/assignment_feedback",
        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
    assert data and "error" in data
```

Case: *Missing required field: user_id*

Request Method: `POST`

Inputs:

```
{
  "code": "print('Hello')"
}
```

Expected Output:

```
HTTP Status Code: 400 and error message about missing user_id
```

Actual Output:

```
HTTP Status Code: 400
JSON: {"message": {"user_id": "User ID is required"}}
```

Result: `Success`

Pytest Code:

```
def test_missing_fields(client):
    payload = {
        "code": "print('Hello')"
        # user_id is missing
    }

    response = client.post("/assignment_feedback", json=payload)
    data = response.get_json()

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

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

    assert response.status_code == 400
    assert data and "message" in data and "user_id" in str(data["message"])
```
