Endpoint Being Tested: http://127.0.0.1:5000/api/handbook/query

Case: *Successful handbook query with relevant answer*

Request Method: POST

Inputs:

```
{
  "query": "What are the system requirements to attend the programme?",
  "k": 1,
  "score_threshold": 0.6
}
```

Expected Output:

```
HTTP Status Code: 200 and JSON with 'answer' and 'documents'
```

Actual Output:


```
HTTP Status Code: 200
JSON: {"answer": "To attend the programme, you need:\n\n* **Hardware:** 8GB RAM
(more is desirable), Intel 8th Gen or AMD 4th Gen (or equivalent) processor, 500GB
storage (1TB desirable, SSD preferred), 13\" laptop or 15\" desktop screen with
1080p resolution, webcam, microphone, and speakers (or headset with mic).\n\n*
**Software:** Windows 10, Ubuntu 20.04 (or equivalent), or macOS Mojave (dual-boot
capability desirable; otherwise, virtualization software like VirtualBox to
emulate Ubuntu 20.04 is needed), latest Chrome browser with a Google account,
familiarity with Google Suite (Docs, Sheets, Slides), and any other software
specified in the course.\n\n* **Internet:** Minimum 2 Mbps connection (higher
bandwidth recommended).\n\n* **Online Interactions/Exams:** Latest Chrome browser
(desirable), mobile device with front camera and good internet (VOLTE ideal),
microphone, and speakers.  All students must pass a system compatibility test.",
"documents": [{"content": "1. RAM Size - 8 GB or higher (The ability to install
more memory is desirable).\n2. Processor - Intel 8th Generation or AMD 4th
Generation or upwards or its equivalent\n(The latest processor configuration is
always recommended)\n3. Storage - Minimum of 500 GB, Desirable of 1 TB. Having an
SSD storage is desirable.\n4. Screen size and resolution - Minimum of 13\u201d for
laptop and Minimum of 15\u201d for desktop\nwith 1080p\n5. Webcamera, a mic and
speaker or an earphone/headphone with mic.\nSoftware/Applications\n1. Operating
System - Minimum requirement of Windows 10 or Ubuntu LTS Version\n20.04 (or any
equivalent) or Mac OS Mojave. Having the capability of dual boot is\ndesirable. If
there is no capability for dual boot, then the operating system must\nsupport
virtualization software like VirtualBox using which Ubuntu 20.04 can be\nemulated.
The System Commands course in Semester 4 will be taught primarily on\nUbuntu
20.04.\n2. Browser - Latest version of Chrome with Google Account signed in\n3.
```

```
Basic familiarity with Google Suite of tools (Docs, Sheets and Slides),
specifically\ncollaboration features.\n4. Any other software that is specified
within the course\nInternet Bandwidth\nMinimum of 2 MBPS connection is required to
attend sessions without disruptions. However\nwe strongly recommend broadband
connections with much higher bandwidth for the best\nlearning experience.\nFor
Online Interactions/Proctored Examinations\n1. Browser - Latest version of Chrome
is desirable\n2. Mobile with a front camera and good internet connection (VOLTE
connections are\nideal)\n3. Mic and speaker to be able to speak to and listen to
the person at the other end\n4. Any applications as required to be installed for
the interactions/examinations\nSystem compatibility test\nAll students have to
mandatorily attend the system compatibility test and ensure that the\nsystem you
have conforms to the above requirements and student has to participate in this
on\nthe dates mentioned by the Admin team to get this completed.\nOther
References\nSome more helpful links that will help you in understanding possible
system configurations:\n", "source": "IITM BS Student Handbook", "page_number":
44}]}
```


Result: Success

Pytest Code:

```python
def test_query_success(client):
    payload = {
        "query": "What are the system requirements to attend the programme?",
        "k": 1,
        "score_threshold": 0.6
    }

    response = client.post("/api/handbook/query", json=payload)
    data = response.get_json()

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

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

    assert response.status_code == 200
    assert "answer" in data
    assert "documents" in data
```

Case: *Missing required 'query' field in payload*

Request Method: POST

Inputs:

```
{
  "k": 1,
  "score_threshold": 0.6
}
```

Expected Output:

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

Actual Output:

```
HTTP Status Code: 400
JSON: {"error": "Query is required"}
```

Result: Success

Pytest Code:

```
def test_query_missing_query_field(client):
    payload = {
        "k": 1,
        "score_threshold": 0.6
    }

    response = client.post("/api/handbook/query", json=payload)
    data = response.get_json()

    expected_status = 400
    result = "Success" if response.status_code == expected_status and "error" in
data else "Failed"

    write_test_doc(
        title="***Case:*** *Missing required 'query' field in payload*",
        endpoint="http://127.0.0.1:5000/api/handbook/query",
        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: *Internal server error on handbook query processing*

Request Method: POST

Inputs:

```
{
  "query": null
}
```

Expected Output:

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

Actual Output:

```
HTTP Status Code: 500
JSON: {"error": "Error processing query: 'NoneType' object has no attribute
'replace'"}
```

Result: Success

Pytest Code:

```
def test_query_internal_server_error(client):
    # This test assumes something goes wrong internally, you can mock or simulate
if needed
    payload = {
        "query": None  # Intentionally passing invalid query to simulate server
error
    }

    response = client.post("/api/handbook/query", 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 handbook query processing*",
        endpoint="http://127.0.0.1:5000/api/handbook/query",
        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
```
