```
            "python",
            "fastapi",
            "book",
            "launch"
        ],
        "location": "Google Meet",
    }

    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {access_token}"
    }

    test_response = {
        "message": "Event created successfully"
    }

    response = await default_client.post("/event/new",
    json=payload, headers=headers)

    assert response.status_code == 200
    assert response.json() == test_response
```

Let's rerun the test file:

```
(venv)$ pytest tests/test_routes.py
```

The result looks like so:

```
pytest
(venv) ➜  planner git:(main) ✗ pytest tests/test_routes.py
========================================== test session starts ==========================================
platform darwin -- Python 3.8.2, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/youngestdev/Work/Building-Web-APIs-with-FastAPI-and-Python/ch08/planner, configfile: pytest.ini
plugins: asyncio-0.18.3, anyio-3.5.0
asyncio: mode=auto
collected 3 items

tests/test_routes.py ...                                                                          [100%]

========================================== 3 passed in 0.04s ==========================================
(venv) ➜  planner git:(main) ✗
```

Figure 8.8 – Successful POST request test run

Let's write a test to verify the count of events stored in the database (in our case, 2). Add the following:

```
@pytest.mark.asyncio
async def test_get_events_count(default_client: httpx.
AsyncClient) -> None:
    response = await default_client.get("/event/")

    events = response.json()

    assert response.status_code == 200
    assert len(events) == 2
```

In the preceding code block, we have stored the JSON response in the `events` variable, whose length is used for our test comparison. Let's rerun the test file:

```
(venv)$ pytest tests/test_routes.py
```

Here's the result:

```
(venv) ➜  planner git:(main) ✗ pytest tests/test_routes.py
========================================== test session starts ==========================================
platform darwin -- Python 3.8.2, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/youngestdev/Work/Building-Web-APIs-with-FastAPI-and-Python/ch08/planner, configfile: pytest.ini
plugins: asyncio-0.18.3, anyio-3.5.0
asyncio: mode=auto
collected 4 items

tests/test_routes.py ....                                                                          [100%]

========================================== 4 passed in 0.04s ==========================================
(venv) ➜  planner git:(main) ✗
```

Figure 8.9 – Successful test run to confirm events count

We have successfully tested the `GET` endpoints `/event` and `/event/{id}` and the `POST` endpoint `/event/new`, respectively. Let's test the `UPDATE` and `DELETE` endpoints for `/event/new` next.

Testing the UPDATE endpoint

Let's start with the `UPDATE` endpoint:

```
@pytest.mark.asyncio
async def test_update_event(default_client: httpx.AsyncClient,
mock_event: Event, access_token: str) -> None:
    test_payload = {
        "title": "Updated FastAPI event"
    }

    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {access_token}"
    }
```

```
    url = f"/event/{str(mock_event.id)}"

    response = await default_client.put(url,
    json=test_payload, headers=headers)

    assert response.status_code == 200
    assert response.json()["title"] ==
    test_payload["title"]
```

In the preceding code block, we are modifying the event stored in the database by retrieving the ID from the `mock_event` fixture. We then define the request payload and the headers. In the `response` variable, the request is initiated and the response retrieved is compared. Let's confirm that the test runs correctly:

```
(venv)$ pytest tests/test_routes.py
```

Here's the result:

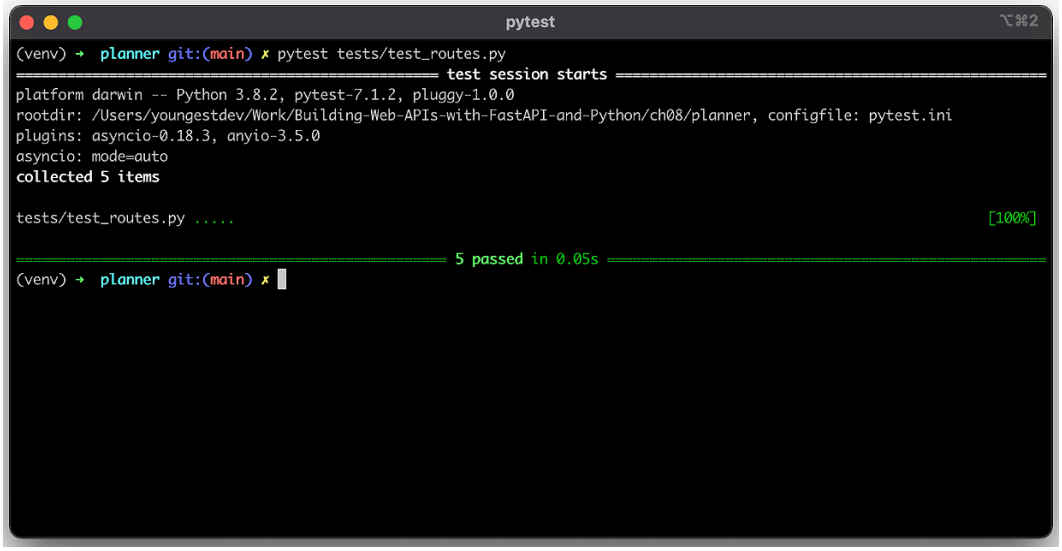

Figure 8.10 – Successful run for UPDATE request

> **Tip**
>
> The `mock_event` fixture comes in handy as the ID for MongoDB documents is uniquely generated every time a document is added to the database.

Let's change the expected response to confirm the validity of our test:

```
    assert response.json()["title"] == "This test should
    fail"
```

Rerun the test:

```
(venv)$ pytest tests/test_routes.py
```

Here's the result:

```
pytest                                                                ⌥⌘2
        test_payload = {
            "title": "Updated FastAPI event"
        }

        headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {access_token}"
        }

        url = f"/event/{str(mock_event.id)}"

        response = await default_client.put(url, json=test_payload, headers=headers)

        assert response.status_code == 200
>       assert response.json()["title"] == "This test should fail"
E       AssertionError: assert 'Updated FastAPI event' == 'This test should fail'
E         - This test should fail
E         + Updated FastAPI event

tests/test_routes.py:103: AssertionError
========================================== short test summary info ==========================================
FAILED tests/test_routes.py::test_update_event - AssertionError: assert 'Updated FastAPI event' == 'This test should fail'
========================================== 1 failed, 4 passed in 0.09s ==========================================
(venv) ➜  planner git:(main) ✗
```

Figure 8.11 – Failed test due to difference in response objects

Testing the DELETE endpoint

Lastly, let's write the test function for the `DELETE` endpoint:

```
@pytest.mark.asyncio
async def test_delete_event(default_client: httpx.AsyncClient,
mock_event: Event, access_token: str) -> None:
    test_response = {
        "message": "Event deleted successfully."
    }

    headers = {
        "Content-Type": "application/json",
```