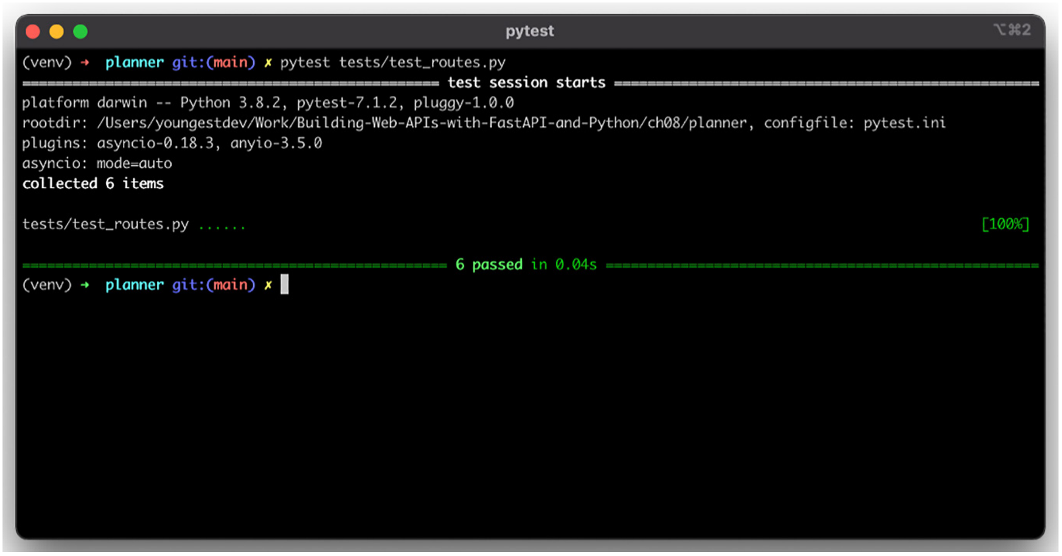
```
            "Authorization": f"Bearer {access_token}"
        }

        url = f"/event/{mock_event.id}"

        response = await default_client.delete(url,
        headers=headers)

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

Like the preceding tests, the expected test response is defined as well as the headers. The `DELETE` route is engaged and the response is compared. Let's run the test:

```
(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 6 items

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

================================= 6 passed in 0.04s =================================
(venv) ➜  planner git:(main) ✗
```

Figure 8.12 – Successful DELETE test

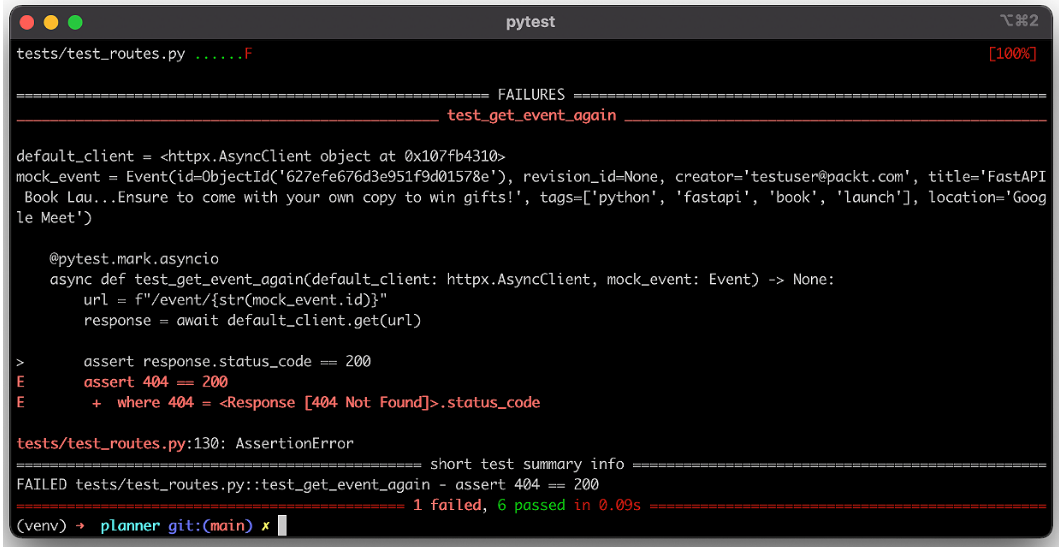
To confirm that the document has indeed been deleted, let's add a final test:

```
@pytest.mark.asyncio
async def test_get_event_again(default_client: httpx.
AsyncClient, mock_event: Event) -> None:
    url = f"/event/{str(mock_event.id)}"
    response = await default_client.get(url)

    assert response.status_code == 200
    assert response.json()["creator"] == mock_event.creator
    assert response.json()["_id"] == str(mock_event.id)
```

The expected response is failure. Let's try it out:

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

Here's the result:



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

===================================== FAILURES =====================================
_______________________________ test_get_event_again _______________________________

default_client = <httpx.AsyncClient object at 0x107fb4310>
mock_event = Event(id=ObjectId('627efe676d3e951f9d01578e'), revision_id=None, creator='testuser@packt.com', title='FastAPI
 Book Lau...Ensure to come with your own copy to win gifts!', tags=['python', 'fastapi', 'book', 'launch'], location='Goog
le Meet')

    @pytest.mark.asyncio
    async def test_get_event_again(default_client: httpx.AsyncClient, mock_event: Event) -> None:
        url = f"/event/{str(mock_event.id)}"
        response = await default_client.get(url)

>       assert response.status_code == 200
E       assert 404 == 200
E        +  where 404 = <Response [404 Not Found]>.status_code

tests/test_routes.py:130: AssertionError
============================== short test summary info ==============================
FAILED tests/test_routes.py::test_get_event_again - assert 404 == 200
============================= 1 failed, 6 passed in 0.09s =============================
(venv) ➜  planner git:(main) ✗ 
```

Figure 8.13 – Failed test response

As seen from the preceding screenshot, the item can no longer be found in the database. Now that you have successfully implemented the tests for authentication and event routes, uncomment the code responsible for clearing out user data from the database:

```
        await User.find_all().delete()
```

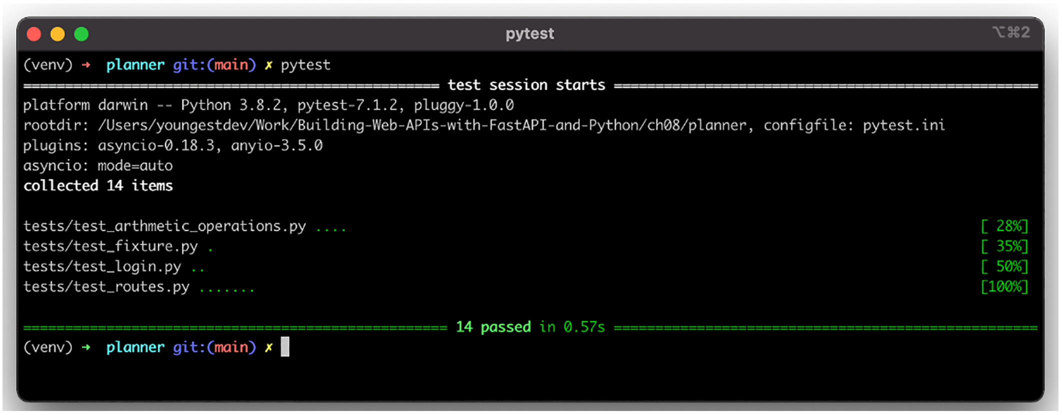
Update the last test:

```
    assert response.status_code == 404
```

Lastly, let's run all the tests present in our application:

```
(venv)$ pytest
```

Here's the result:



```
pytest                                                                          ⌥⌘2
(venv) ➜  planner git:(main) ✗ pytest
================================================== 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 14 items

tests/test_arthmetic_operations.py ....                                                                          [ 28%]
tests/test_fixture.py .                                                                                          [ 35%]
tests/test_login.py ..                                                                                           [ 50%]
tests/test_routes.py .......                                                                                     [100%]

=================================================== 14 passed in 0.57s ==================================================
(venv) ➜  planner git:(main) ✗ 
```

Figure 8.14 – Complete tests ran in 0.57 seconds

Now that we have successfully tested the endpoints contained in the event-planner API, let's run a coverage test to determine the percentage of our code involved in the test operation.

Test coverage

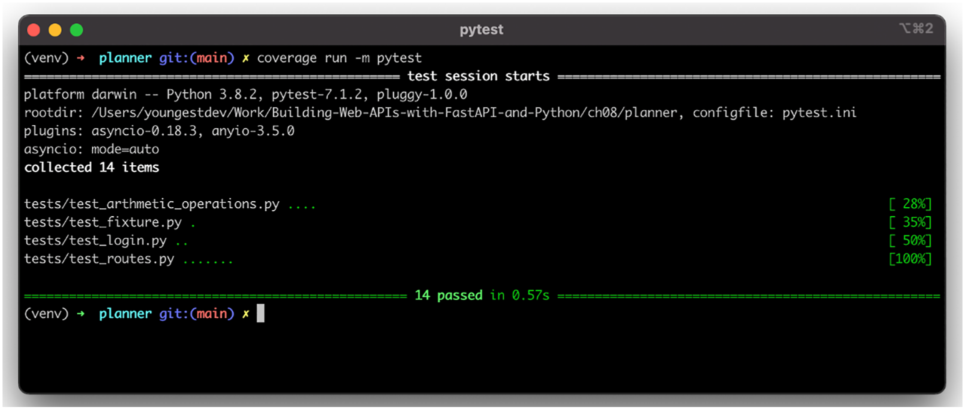
A test coverage report is useful in determining the percentage of our code that was executed in the course of testing. Let's install the `coverage` module so we can measure whether our API was adequately tested:

```
(venv)$ pip install coverage
```

Next, let's generate a coverage report by running this command:

```
(venv)$ coverage run -m pytest
```

Here's the result:



```
(venv) ➜  planner git:(main) ✗ coverage run -m pytest
================================================== 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 14 items

tests/test_arthmetic_operations.py ....                                                                           [ 28%]
tests/test_fixture.py .                                                                                           [ 35%]
tests/test_login.py ..                                                                                            [ 50%]
tests/test_routes.py .......                                                                                      [100%]

=================================================== 14 passed in 0.57s ===================================================
(venv) ➜  planner git:(main) ✗ 
```

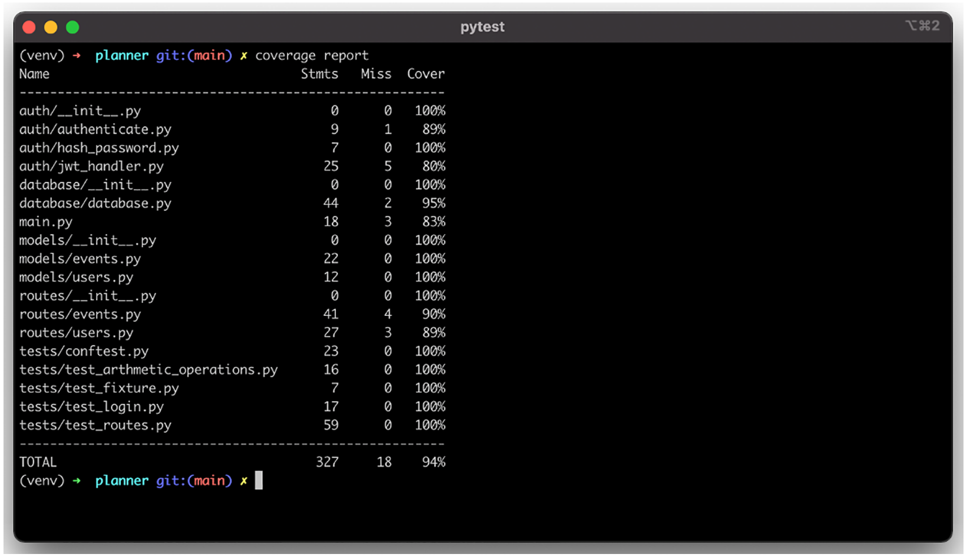
Figure 8.15 – Coverage report generated

Next, let's view the report generated by the `coverage run -m pytest` command. We can choose to view the report on the terminal or a web page by generating an HTML report. We'll do both.

Let's review the report from the terminal:

```
(venv)$ coverage report
```

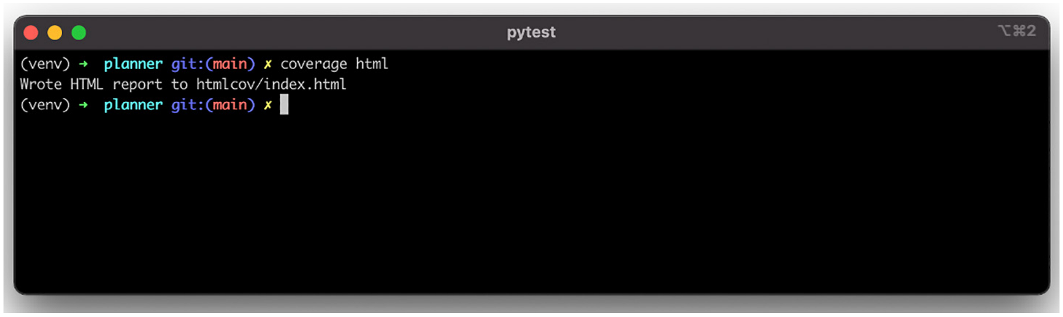
Here's the result:



```
(venv) ➜  planner git:(main) ✗ coverage report
Name                                    Stmts   Miss  Cover
-----------------------------------------------------------
auth/__init__.py                            0      0   100%
auth/authenticate.py                        9      1    89%
auth/hash_password.py                       7      0   100%
auth/jwt_handler.py                        25      5    80%
database/__init__.py                        0      0   100%
database/database.py                       44      2    95%
main.py                                    18      3    83%
models/__init__.py                          0      0   100%
models/events.py                           22      0   100%
models/users.py                            12      0   100%
routes/__init__.py                          0      0   100%
routes/events.py                           41      4    90%
routes/users.py                            27      3    89%
tests/conftest.py                          23      0   100%
tests/test_arthmetic_operations.py         16      0   100%
tests/test_fixture.py                       7      0   100%
tests/test_login.py                        17      0   100%
tests/test_routes.py                       59      0   100%
-----------------------------------------------------------
TOTAL                                     327     18    94%
(venv) ➜  planner git:(main) ✗ 
```

Figure 8.16 – Coverage report from the terminal

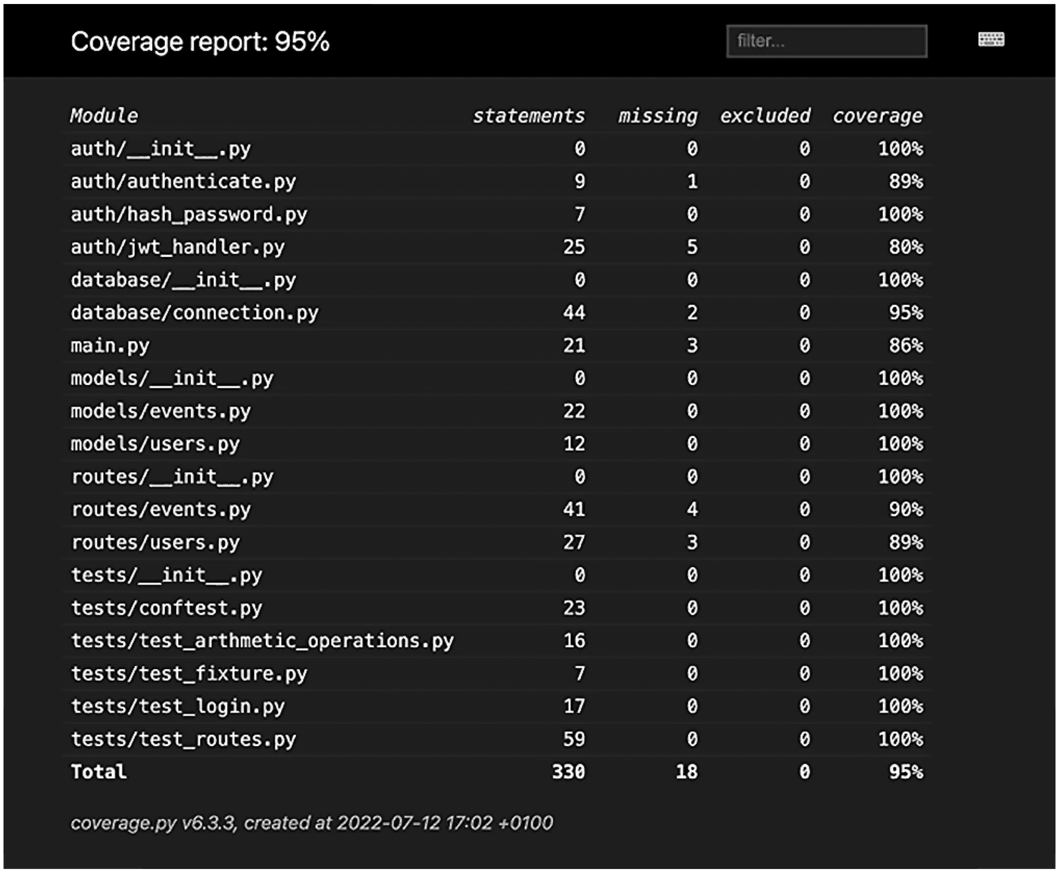
From the preceding report, the percentages signify the amount of code executed and interacted with. Let's generate the HTML report so we can check the blocks of code interacted with.



```
(venv) ➜  planner git:(main) ✗ coverage html
Wrote HTML report to htmlcov/index.html
(venv) ➜  planner git:(main) ✗
```

Figure 8.17 – Generating an HTML coverage report

Next, open `htmlcov/index.html` from your browser.



Coverage report: 95%

Module	statements	missing	excluded	coverage
auth/__init__.py	0	0	0	100%
auth/authenticate.py	9	1	0	89%
auth/hash_password.py	7	0	0	100%
auth/jwt_handler.py	25	5	0	80%
database/__init__.py	0	0	0	100%
database/connection.py	44	2	0	95%
main.py	21	3	0	86%
models/__init__.py	0	0	0	100%
models/events.py	22	0	0	100%
models/users.py	12	0	0	100%
routes/__init__.py	0	0	0	100%
routes/events.py	41	4	0	90%
routes/users.py	27	3	0	89%
tests/__init__.py	0	0	0	100%
tests/conftest.py	23	0	0	100%
tests/test_arthmetic_operations.py	16	0	0	100%
tests/test_fixture.py	7	0	0	100%
tests/test_login.py	17	0	0	100%
tests/test_routes.py	59	0	0	100%
Total	**330**	**18**	**0**	**95%**

coverage.py v6.3.3, created at 2022-07-12 17:02 +0100

Figure 8.18 – Coverage report from the web browser