テスト

Starlette のおかげで、**FastAPI** アプリケーションのテストは簡単で楽しいものになっています。

Requests がベースなので、非常に使いやすく直感的です。

これを使用すると、**FastAPI** と共に pytest を直接利用できます。

`TestClient` を使用

`TestClient` をインポートします。

`TestClient` を作成し、**FastAPI** に渡します。

`test_` から始まる名前の関数を作成します (これは `pytest` の標準的なコンベンションです)。

`requests` と同じ様に `TestClient` オブジェクトを使用します。

チェックしたい Python の標準的な式と共に、シンプルに `assert` 文を記述します。

```
{!../../../docs_src/app_testing/tutorial001.py!}
```

!!! tip "豆知識" テスト関数は `async def` ではなく、通常の `def` であることに注意してください。

また、クライアントへの呼び出しも通常の呼び出しであり、`await` を使用しません。

これにより、煩雑にならずに、`pytest` を直接使用できます。

!!! note "技術詳細" `from starlette.testclient import TestClient` も使用できます。

`**FastAPI**` は開発者の利便性のために `fastapi.testclient` と同じ `starlette.testclient` を提供します。しかし、実際にはStarletteから直接渡されています。

!!! tip "豆知識" FastAPIアプリケーションへのリクエストの送信とは別に、テストで `async` 関数 (非同期データベース関数など) を呼び出したい場合は、高度なチュートリアルのAsync Tests{.internal-link target=_blank} を参照してください。

テストの分離

実際のアプリケーションでは、おそらくテストを別のファイルに保存します。

また、**FastAPI** アプリケーションは、複数のファイル/モジュールなどで構成されている場合もあります。

FastAPI アプリファイル

FastAPI アプリに `main.py` ファイルがあるとします:

```
{!../../../docs_src/app_testing/main.py!}
```

テストファイル

次に、テストを含む `test_main.py` ファイルを作成し、 `main` モジュール (`main.py`) から `app` をインポートします:

```
{!../../../docs_src/app_testing/test_main.py!}
```

テスト: 例の拡張

次に、この例を拡張し、詳細を追加して、さまざまなパーツをテストする方法を確認しましょう。

拡張版 FastAPI アプリファイル

FastAPI アプリに `main_b.py` ファイルがあるとします。

そのファイルには、エラーを返す可能性のある `GET` オペレーションがあります。

また、いくつかのエラーを返す可能性のある `POST` オペレーションもあります。

これらの *path operation* には `X-Token` ヘッダーが必要です。

```
{!../../../docs_src/app_testing/main_b.py!}
```

拡張版テストファイル

次に、先程のものに拡張版のテストを加えた、 `test_main_b.py` を作成します。

```
{!../../../docs_src/app_testing/test_main_b.py!}
```

リクエストに情報を渡せるクライアントが必要で、その方法がわからない場合はいつでも、 `requests` での実現方法を検索 (Google) できます。

テストでも同じことを行います。

例えば:

- *パス* または *クエリ* パラメータを渡すには、それをURL自体に追加します。
- JSONボディを渡すには、Pythonオブジェクト (例: `dict`) を `json` パラメータに渡します。
- JSONの代わりに *フォームデータ* を送信する必要がある場合は、代わりに `data` パラメータを使用してください。
- ヘッダー を渡すには、 `headers` パラメータに `dict` を渡します。
- *cookies* の場合、 `cookies` パラメータに `dict` です。

(`requests` または `TestClient` を使用して) バックエンドにデータを渡す方法の詳細は、Requestsのドキュメントを確認してください。

!!! info "情報" `TestClient` は、Pydanticモデルではなく、JSONに変換できるデータを受け取ることに注意してください。

テストにPydanticモデルがあり、テスト中にそのデータをアプリケーションに送信したい場合は、`[JSON互換エンコーダ](encoder.md){.internal-link target=_blank}` で説明されている `jsonable_encoder` が利用できます。

実行

後は、 `pytest` をインストールするだけです:

```
$ pip install pytest

---> 100%
```

ファイルを検知し、自動テストを実行し、結果のレポートを返します。

以下でテストを実行します:

```
$ pytest

================ test session starts ================
platform linux -- Python 3.6.9, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /home/user/code/superawesome-cli/app
plugins: forked-1.1.3, xdist-1.31.0, cov-2.8.1
collected 6 items

---> 100%

test_main.py <span style="color: green; white-space: pre;">......
[100%]</span>

<span style="color: green;">================= 1 passed in 0.03s =================
</span>
```