



FastAPI framework, high performance, easy to learn, fast to code, ready for production

build passing coverage 100% pypi package 0.81.0

ドキュメント: <https://fastapi.tiangolo.com>

ソースコード: <https://github.com/tiangolo/fastapi>



FastAPI は、Pythonの標準である型ヒントに基づいてPython 3.6 以降でAPI を構築するための、モダンで、高速(高パフォーマンス)な、Web フレームワークです。

主な特徴:

- **高速:** NodeJS や Go 並みのとても高いパフォーマンス (Starlette と Pydanticのおかげです)。 [最も高速な Python フレームワークの一つです](#)。
- **高速なコーディング:** 開発速度を約 200%~300%向上させます。 *
- **少ないバグ:** 開発者起因のヒューマンエラーを約 40%削減します。 *
- **直感的:** 素晴らしいエディタのサポートや オートコンプリート... デバッグ時間を削減します。
- **簡単:** 簡単に利用、習得できるようにデザインされています。ドキュメントを読む時間を削減します。
- **短い:** コードの重複を最小限にしています。各パラメータからの複数の機能。少ないバグ。
- **堅牢性:** 自動対話ドキュメントを使用して、本番環境で利用できるコードを取得します。
- **Standards-based:** API のオープンスタンダードに基づいており、完全に互換性があります: [OpenAPI](#) (以前は Swagger として知られていました) や [JSON スキーマ](#)。

* 本番アプリケーションを構築している開発チームのテストによる見積もり。

Sponsors

{% if sponsors %} {% for sponsor in sponsors.gold -%}  {% endfor -%} {%- for sponsor in sponsors.silver -%}  {% endfor %} {% endif %}

[Other sponsors](#)

評価

"[...] 最近 **FastAPI** を使っています。[...] 実際に私のチームの全ての **Microsoft の機械学習サービス** で使用する予定です。 そのうちのいくつかのコアな**Windows製品**と**Office製品**に統合されつつあります。 "

Kabir Khan - **Microsoft** ([ref](#))

"FastAPI ライブラリを採用し、クエリで**予測値**を取得できる**REST**サーバを構築しました。 [for Ludwig]"

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala - **Uber** ([ref](#))

"**Netflix** は、**危機管理**オーケストレーションフレームワーク、**Dispatch** のオープンソースリリースを発表できることをうれしく思います。 [built with **FastAPI**]"

Kevin Glisson, Marc Vilanova, Forest Monsen - **Netflix** ([ref](#))

"私は**FastAPI**にワクワクしています。 めちゃくちゃ楽しいです！"

Brian Okken - **Python Bytes** podcast host ([ref](#))

"正直、**超堅実**で洗練されているように見えます。 いろんな意味で、それは私がハグしたかったものです。 "

Timothy Crosley - **Hug**, creator ([ref](#))

"REST API を構築するための**モダンなフレームワーク**を学びたい方は、**FastAPI** [...] をチェックしてみてください。 [...] 高速で、使用、習得が簡単です。 [...]"

"私たちの**API**は**FastAPI**に切り替えました。 [...] きっと気に入ると思います。 [...]"

Ines Montani - Matthew Honnibal - **Explosion AI** founders - **spaCy**, creators ([ref](#)) - ([ref](#))

Typer, the FastAPI of CLIs



もし Web API の代わりにターミナルで使用する**CLI**アプリを構築する場合は、**Typer**を確認してください。

Typerは FastAPI の弟分です。そして、**CLI 版 の FastAPI**を意味しています。

必要条件

Python 3.6+

FastAPI は巨人の肩の上に立っています。

- Web の部分は[Starlette](#)
- データの部分は[Pydantic](#)

インストール

```
$ pip install fastapi
```

```
---> 100%
```

本番環境では、[Uvicorn](#) または、[Hypercorn](#) のような、ASGI サーバーが必要になります。

```
$ pip install uvicorn[standard]
```

```
---> 100%
```

アプリケーション例

アプリケーションの作成

- `main.py` を作成し、以下のコードを入力します:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

- ▶ または `async def` を使います...

実行

以下のコマンドでサーバーを起動します:

```
$ uvicorn main:app --reload
```

```
INFO:      Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:      Started reloader process [28720]
INFO:      Started server process [28722]
INFO:      Waiting for application startup.
INFO:      Application startup complete.
```

- ▶ `uvicorn main:app --reload` コマンドについて

動作確認

ブラウザから <http://127.0.0.1:8000/items/5?q=somequery> を開きます。

以下の JSON のレスポンスが確認できます:

```
{"item_id": 5, "q": "somequery"}
```

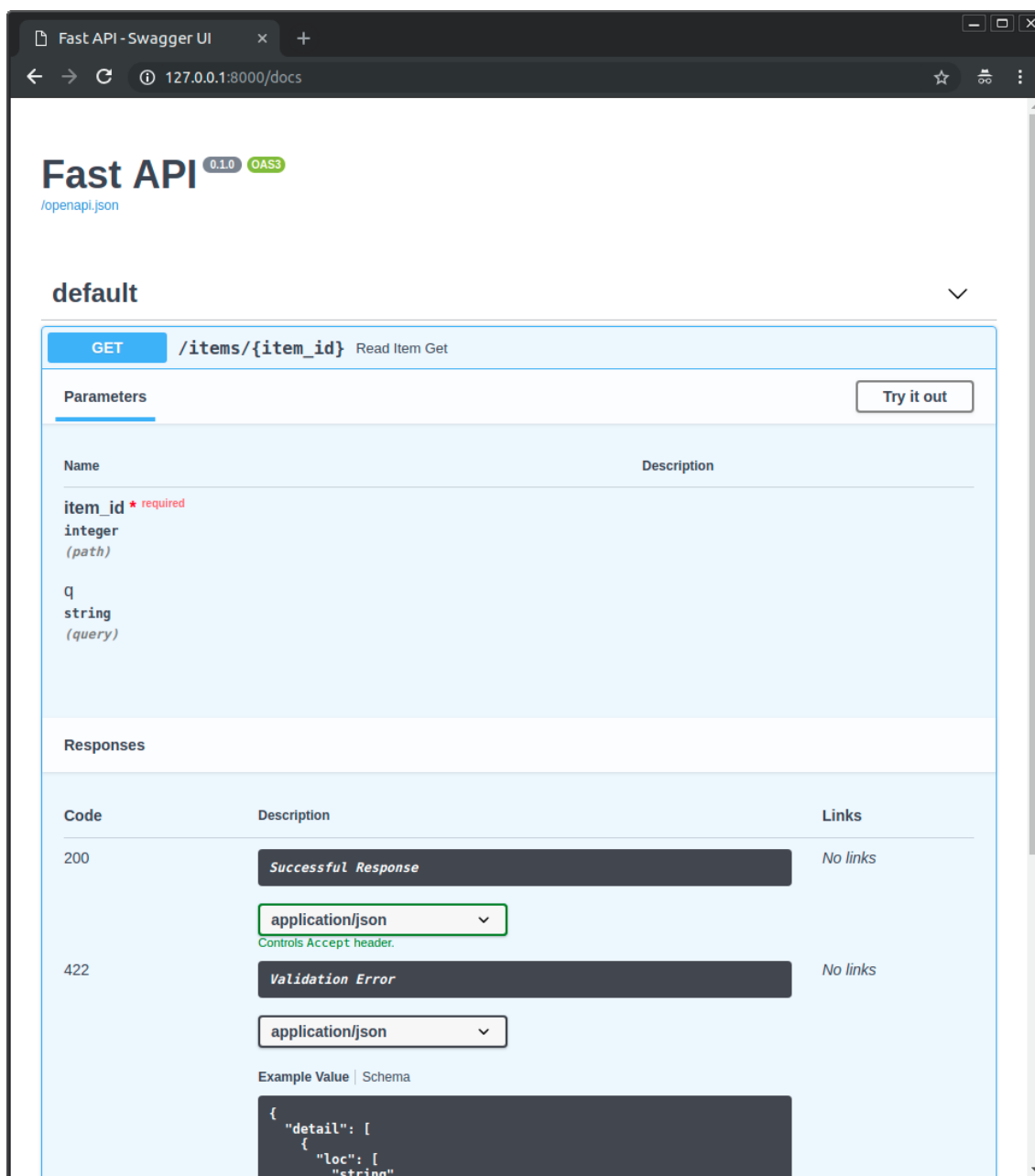
もうすでに以下の API が作成されています:

- `/` と `/items/{item_id}` のパスで HTTP リクエストを受けます。
- どちらのパスも `GET` 操作を取ります。(HTTP メソッドとしても知られています。)
- `/items/{item_id}` パスのパスパラメータ `item_id` は `int` でなければなりません。
- パス `/items/{item_id}` はオプションの `str` クエリパラメータ `q` を持ちます。

自動対話型の API ドキュメント

<http://127.0.0.1:8000/docs>にアクセスしてみてください。

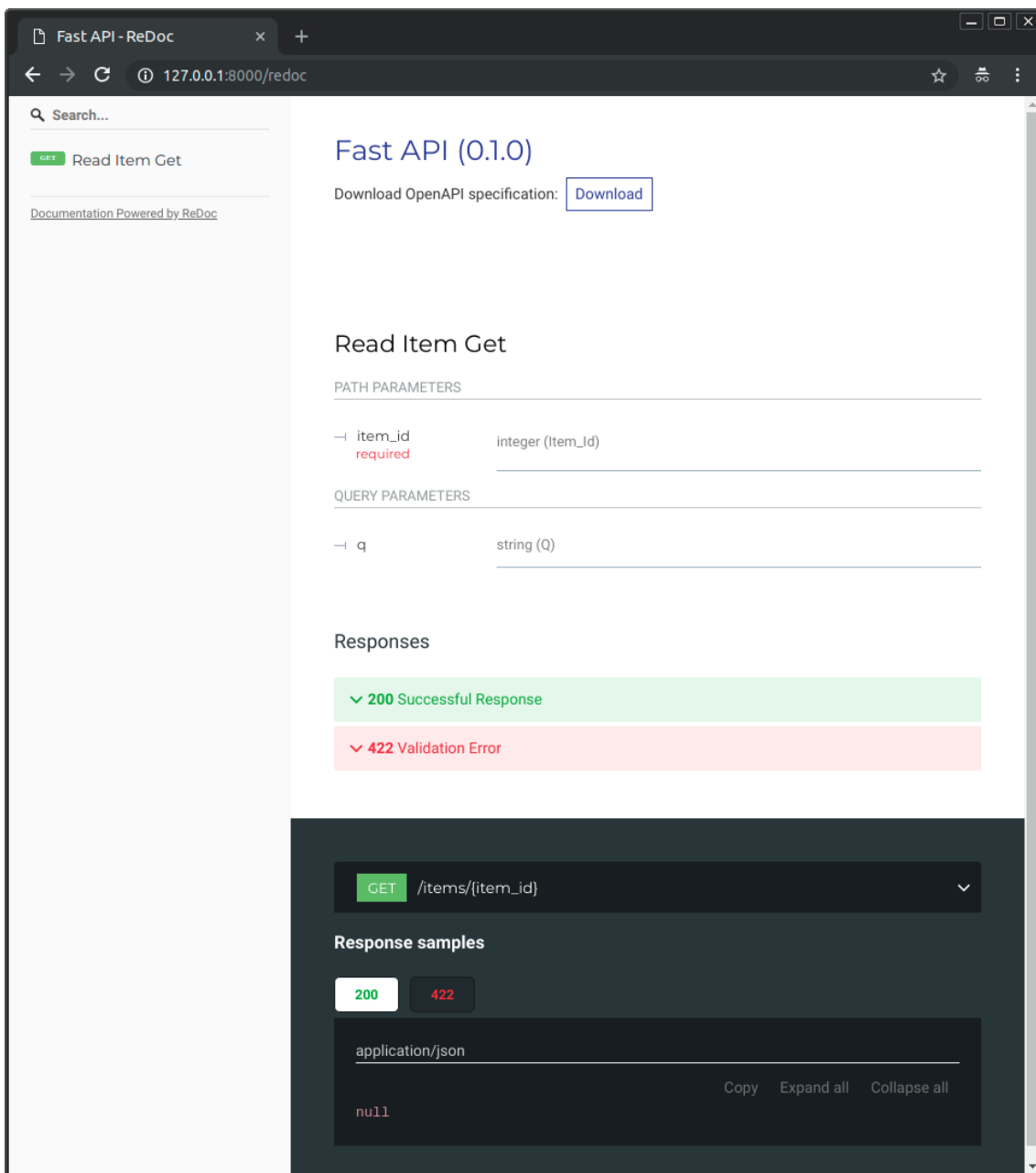
自動対話型の API ドキュメントが表示されます。([Swagger UI](#) が提供しています。):



代替の API ドキュメント

<http://127.0.0.1:8000/redoc>にアクセスしてみてください。

代替の自動ドキュメントが表示されます。(ReDocが提供しています。):



アップグレード例

PUT リクエストからボディを受け取るために `main.py` を修正しましょう。

Pydantic によって、Python の標準的な型を使ってボディを宣言します。

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
```

```
class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = None

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}
```

サーバーは自動でリロードされます。(上述の `uvicorn` コマンドで `--reload` オプションを追加しているからです。)

自動対話型の API ドキュメントのアップグレード

<http://127.0.0.1:8000/docs> にアクセスしましょう。

- 自動対話型の API ドキュメントが新しいボディも含めて自動でアップデートされます:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Fast API 0.1.0 OAS3

/openapi.json

default

GET / Read Root Get

GET /items/{item_id} Read Item Get

PUT /items/{item_id} Save Item Put

Parameters Try it out

Name Description

item_id * required
integer
(path)

Request body required application/json

Example Value | Schema

```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```

Responses

Code	Description	Links
200	Successful Response	No links

- "Try it out"ボタンをクリックしてください。パラメータを入力して API と直接やりとりすることができます。

Fast API 0.1.0 OAS3
/openapi.json

default

GET / Read Root Get

GET /items/{item_id} Read Item Get

PUT /items/{item_id} Save Item Put

Parameters

Name	Description
item_id * required integer (path)	1234

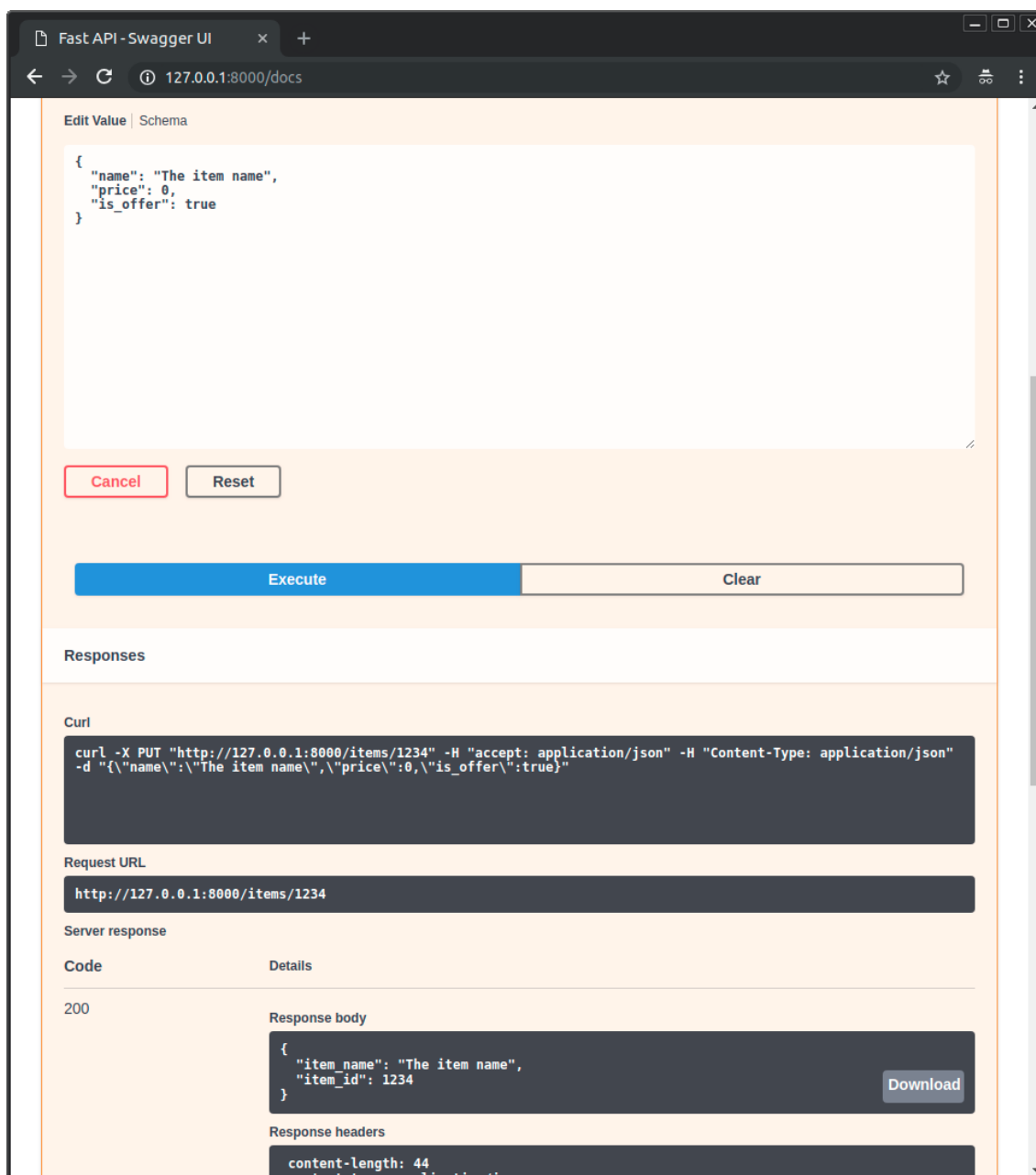
Request body required application/json

Edit Value | Schema

```
{  
  "name": "The item name",  
  "price": 0,  
  "is_offer": true  
}
```

Cancel Reset

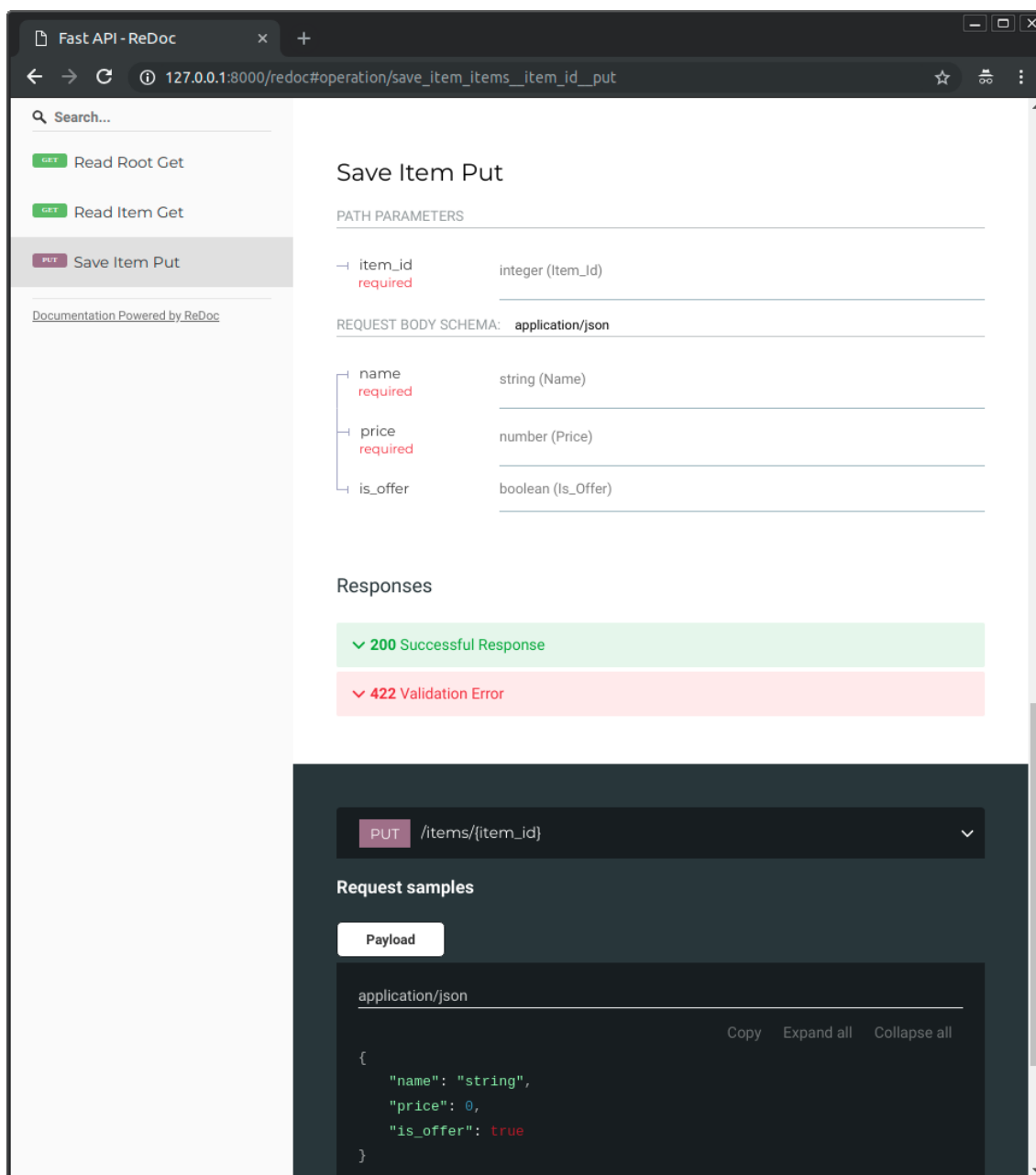
- それから、"Execute" ボタンをクリックしてください。ユーザーインターフェースは API と通信し、パラメータを送信し、結果を取得して画面に表示します:



代替の API ドキュメントのアップグレード

<http://127.0.0.1:8000/redoc>にアクセスしましょう。

- 代替の API ドキュメントにも新しいクエリパラメータやボディが反映されます。



まとめ

要約すると、関数のパラメータとして、パラメータやボディなどの型を**一度だけ**宣言します。

標準的な最新の Python の型を使っています。

新しい構文や特定のライブラリのメソッドやクラスなどを覚える必要はありません。

単なる標準的な**3.6 以降の Python**です。

例えば、`int` の場合:

```
item_id: int
```

または、より複雑な `Item` モデルの場合:

```
item: Item
```

...そして、この一度の宣言で、以下ようになります:

- 以下を含むエディタサポート:
 - 補完
 - タイプチェック
- データの検証:
 - データが無効な場合に自動でエラーをクリアします。
 - 深い入れ子になった JSON オブジェクトでも検証が可能です。
- 入力データの変換: ネットワークから Python のデータや型に変換してから読み取ります:
 - JSON.
 - パスパラメータ
 - クエリパラメータ
 - クッキー
 - ヘッダー
 - フォーム
 - ファイル
- 出力データの変換: Python のデータや型からネットワークデータへ変換します (JSON として):
 - Convert Python types (`str` , `int` , `float` , `bool` , `list` , etc).
 - `datetime` オブジェクト
 - `UUID` オブジェクト
 - データベースモデル
 - ...などなど
- 2つの代替ユーザーインターフェースを含む自動インタラクティブ API ドキュメント:
 - Swagger UI.
 - ReDoc.

コード例に戻りましょう、**FastAPI** は次のようになります:

- GET および PUT リクエストのパスに `item_id` があることを検証します。
- `item_id` が GET および PUT リクエストに対して `int` 型であることを検証します。
 - そうでない場合は、クライアントは有用で明確なエラーが表示されます。
- GET リクエストに対してオプションのクエリパラメータ `q` (`http://127.0.0.1:8000/items/foo?q=somequery` のように) が存在するかどうかを調べます。
 - パラメータ `q` は `= None` で宣言されているので、オプションです。
 - `None` がなければ必須になります (PUT の場合のボディと同様です)。
- PUT リクエストを `/items/{item_id}` に送信する場合は、ボディを JSON として読み込みます:
 - 必須の属性 `name` を確認してください。それは `str` であるべきです。
 - 必須の属性 `price` を確認してください。それは `float` でなければならないです。
 - オプションの属性 `is_offer` を確認してください。値がある場合は、`bool` であるべきです。
 - これらはすべて、深くネストされた JSON オブジェクトに対しても動作します。
- JSON から JSON に自動的に変換します。
- OpenAPIですべてを文書化し、以下を使用することができます:

- 対話的なドキュメントシステム。
- 多くの言語に対応した自動クライアントコード生成システム。
- 2つの対話的なドキュメントのWebインターフェイスを直接提供します。

まだ表面的な部分に触れただけですが、もう全ての仕組みは分かっているはずです。

以下の行を変更してみてください:

```
return {"item_name": item.name, "item_id": item_id}
```

...以下を:

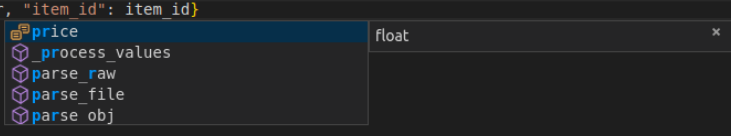
```
... "item_name": item.name ...
```

...以下のように:

```
... "item_price": item.price ...
```

...そして、エディタが属性を自動補完し、そのタイプを知る方法を確認してください。:

```
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3
4  app = FastAPI()
5
6
7  class Item(BaseModel):
8      name: str
9      price: float
10     is_offer: bool = None
11
12
13 @app.get("/")
14 def read_root():
15     return {"Hello": "World"}
16
17
18 @app.get("/items/{item_id}")
19 def read_item(item_id: int, q: str = None):
20     return {"item_id": item_id, "q": q}
21
22
23 @app.put("/items/{item_id}")
24 def save_item(item_id: int, item: Item):
25     return {"item_name": item.pr, "item_id": item_id}
26
```



より多くの機能を含む、より完全な例については、[チュートリアル - ユーザーガイド](#)をご覧ください。

ネタバレ注意: チュートリアル - ユーザーガイドは以下の情報が含まれています:

- ヘッダー、クッキー、フォームフィールド、ファイルなどの他の場所からの **パラメータ 宣言**。
- `maximum_length` や `regex` のような **検証**や**制約**を設定する方法。
- 非常に強力で使いやすい **依存性注入**システム。
- **JWT トークン**を用いた **OAuth2** や **HTTP Basic 認証** のサポートを含む、セキュリティと認証。

- 深くネストされた **JSON モデル**を宣言するためのより高度な（しかし同様に簡単な）技術（Pydantic のおかげです）。
- 以下のようなたくさんのおまけ機能(Starlette のおかげです):
 - **WebSockets**
 - **GraphQL**
 - `requests` や `pytest` をもとにした極限に簡単なテスト
 - **CORS**
 - **クッキーセッション**
 - ...などなど。

パフォーマンス

独立した TechEmpower のベンチマークでは、Uvicorn で動作する **FastAPI**アプリケーションが、[Python フレームワークの中で最も高速なものの 1 つ](#)であり、Starlette と Uvicorn（FastAPI で内部的に使用されています）にのみ下回っていると示されています。

詳細は[ベンチマーク](#)セクションをご覧ください。

オプションの依存関係

Pydantic によって使用されるもの:

- [ujson](#) - より速い JSON への"変換".
- [email_validator](#) - Eメールの検証

Starlette によって使用されるもの:

- [requests](#) - `TestClient` を使用するために必要です。
- [aiofiles](#) - `FileResponse` または `StaticFiles` を使用したい場合は必要です。
- [jinja2](#) - デフォルトのテンプレート設定を使用する場合は必要です。
- [python-multipart](#) - `"parsing"` `request.form()` からの変換をサポートしたい場合は必要です。
- [itsdangerous](#) - `SessionMiddleware` サポートのためには必要です。
- [pyyaml](#) - Starlette の `SchemaGenerator` サポートのために必要です。(FastAPI では必要ないでしょう。)
- [graphene](#) - `GraphQLApp` サポートのためには必要です。
- [ujson](#) - `UJSONResponse` を使用する場合は必須です。

FastAPI / Starlette に使用されるもの:

- [uvicorn](#) - アプリケーションをロードしてサーブするサーバーのため。
- [orjson](#) - `ORJSONResponse` を使用したい場合は必要です。

これらは全て `pip install fastapi[all]` でインストールできます。

ライセンス

このプロジェクトは MIT ライセンスです。