FastAPI 프레임워크, 고성능, 간편한 학습, 빠른 코드 작성, 준비된 프로덕션

문서: https://fastapi.tiangolo.com

소스 코드: https://github.com/tiangolo/fastapi

FastAPI는 현대적이고, 빠르며(고성능), 파이썬 표준 타입 힌트에 기초한 Python3.6+의 API를 빌드하기 위한 웹 프레임워크입니다.

주요 특징으로:

- **빠름**: (Starlette과 Pydantic 덕분에) **NodeJS** 및 **Go**와 대등할 정도로 매우 높은 성능. 사용 가능한 가장 빠른 파이썬 프레임워크 중 하나.
- **빠른 코드 작성**: 약 200%에서 300%까지 기능 개발 속도 증가. *
- **적은 버그**: 사람(개발자)에 의한 에러 약 40% 감소. *
- **직관적**: 훌륭한 편집기 지원. 모든 곳에서 자동완성. 적은 디버깅 시간.
- **쉬움**: 쉽게 사용하고 배우도록 설계. 적은 문서 읽기 시간.
- **짧음**: 코드 중복 최소화. 각 매개변수 선언의 여러 기능. 적은 버그.
- **견고함**: 준비된 프로덕션 용 코드를 얻으십시오. 자동 대화형 문서와 함께.
- **표준 기반**: API에 대한 (완전히 호환되는) 개방형 표준 기반: OpenAPI (이전에 Swagger로 알려졌던) 및 JSON 스키마.

* 내부 개발팀의 프로덕션 애플리케이션을 빌드한 테스트에 근거한 측정

골드 스폰서

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

다른 스폰서

의견들

"*[...] 저는 요즘 **FastAPI**를 많이 사용하고 있습니다. [...] 사실 우리 팀의 **마이크로소프트 ML 서비스** 전부를 바꿀 계획입니다. 그중 일부는 핵심 **Windows**와 몇몇의 **Office** 제품들이 통합되고 있습니다.*"

Kabir Khan - **마이크로소프트** (ref)

"***FastAPI** 라이브러리를 채택하여 **예측**을 얻기 위해 쿼리를 실행 할 수 있는 **REST** 서버를 생성했습니다. [Ludwig을 위해]*"

Piero Molino, Yaroslav Dudin 그리고 Sai Sumanth Miryala - **우버** (ref)

"***Netflix**는 우리의 오픈 소스 배포판인 **위기 관리** 오케스트레이션 프레임워크를 발표할 수 있어 기쁩니다: 바로 **Dispatch**입니다! [**FastAPI**로 빌드]*"

Kevin Glisson, Marc Vilanova, Forest Monsen - **넷플릭스** (ref)

"***FastAPI**가 너무 좋아서 구름 위를 걷는듯 합니다. 정말 즐겁습니다!*"

Brian Okken - **Python Bytes 팟캐스트 호스트** (ref)

"*솔직히, 당신이 만든 것은 매우 견고하고 세련되어 보입니다. 여러 면에서 **Hug**가 이렇게 되었으면 합니다 - 그걸 만든 누군가를 보는 것은 많은 영감을 줍니다.*"

Timothy Crosley - **Hug 제작자** (ref)

"*REST API를 만들기 위해 **현대적인 프레임워크**를 찾고 있다면 **FastAPI**를 확인해 보십시오. [...] 빠르고, 쓰기 쉽고, 배우기도 쉽습니다 [...]*"

"*우리 **API**를 **FastAPI**로 바꿨습니다 [...] 아마 여러분도 좋아하실 것입니다 [...]*"

Ines Montani - Matthew Honnibal - **Explosion AI 설립자 - spaCy 제작자** (ref) - (ref)

Typer, FastAPI의 CLI

웹 API 대신 터미널에서 사용할 CLI 앱을 만들고 있다면, **Typer**를 확인해 보십시오.

Typer는 FastAPI의 동생입니다. 그리고 **FastAPI의 CLI**가 되기 위해 생겼습니다. ⌨ 🚀

요구사항

Python 3.6+

FastAPI는 거인들의 어깨 위에 서 있습니다:

- 웹 부분을 위한 Starlette.
- 데이터 부분을 위한 Pydantic.

설치

```
$ pip install fastapi

---> 100%
```

프로덕션을 위해 Uvicorn 또는 Hypercorn과 같은 ASGI 서버도 필요할 겁니다.

```
$ pip install uvicorn[standard]

---> 100%
```

예제

만들기

- `main.py` 파일을 만드십시오:

```python
from typing import Optional

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: Optional[str] = None):
    return {"item_id": item_id, "q": q}
```

▶ 또는 `async def` 사용하기...

실행하기

서버를 실행하십시오:

```
$ 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` 명령에 관하여...

확인하기

브라우저로 http://127.0.0.1:8000/items/5?q=somequery를 열어보십시오.

아래의 JSON 응답을 볼 수 있습니다:

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

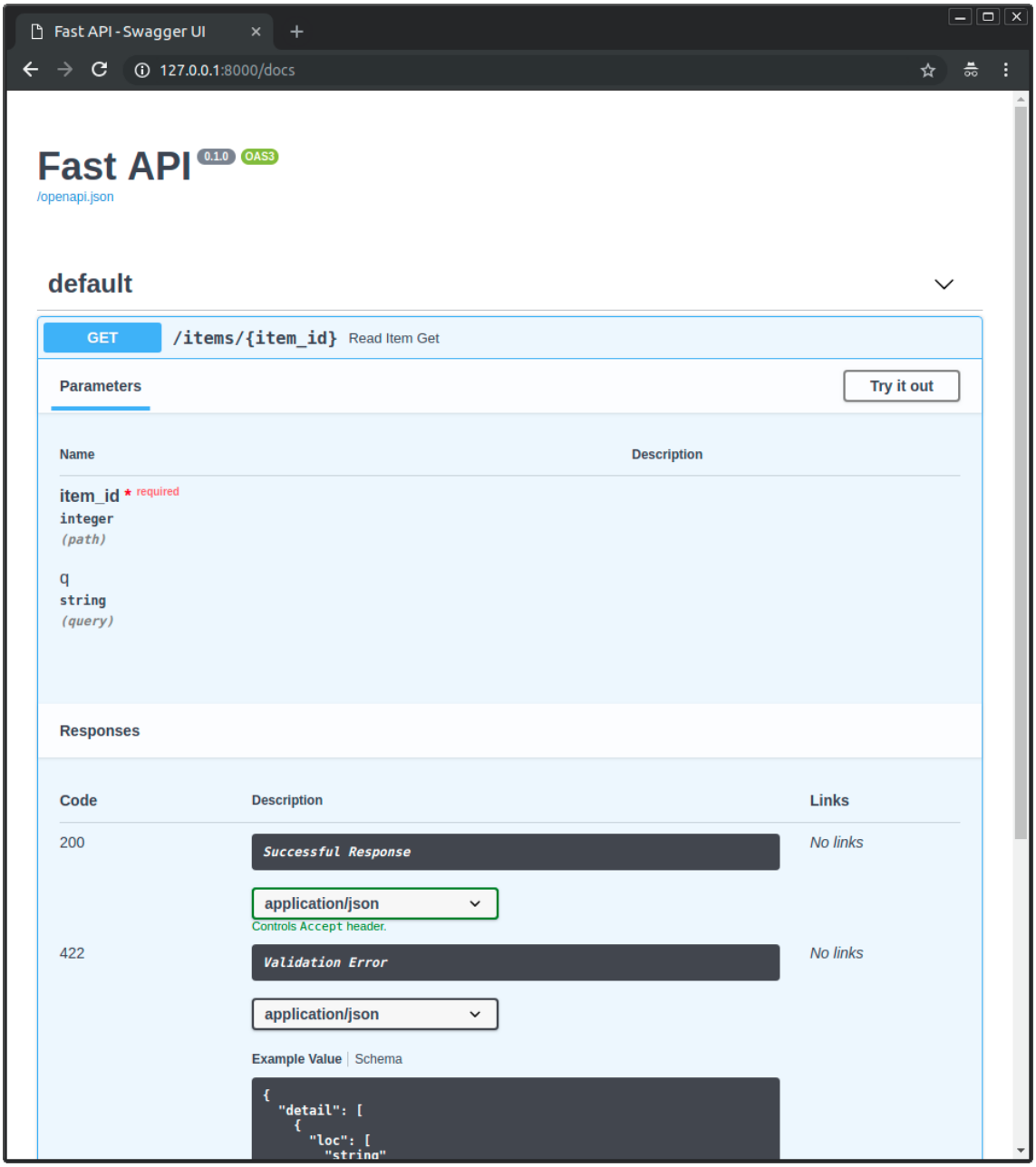
여러분은 벌써 API를 만들었습니다:

- *경로* `/` 및 `/items/{item_id}`에서 HTTP 요청 받기.
- 두 *경로* 모두 `GET` *연산*(HTTP *메소드* 로 알려진)을 받습니다.
- *경로* `/items/{item_id}` 는 *경로 매개변수* `int` 형 이어야 하는 `item_id` 를 가지고 있습니다.
- *경로* `/items/{item_id}` 는 선택적인 `str` 형 이어야 하는 *경로 매개변수* `q` 를 가지고 있습니다.

대화형 API 문서

이제 http://127.0.0.1:8000/docs로 가보십시오.

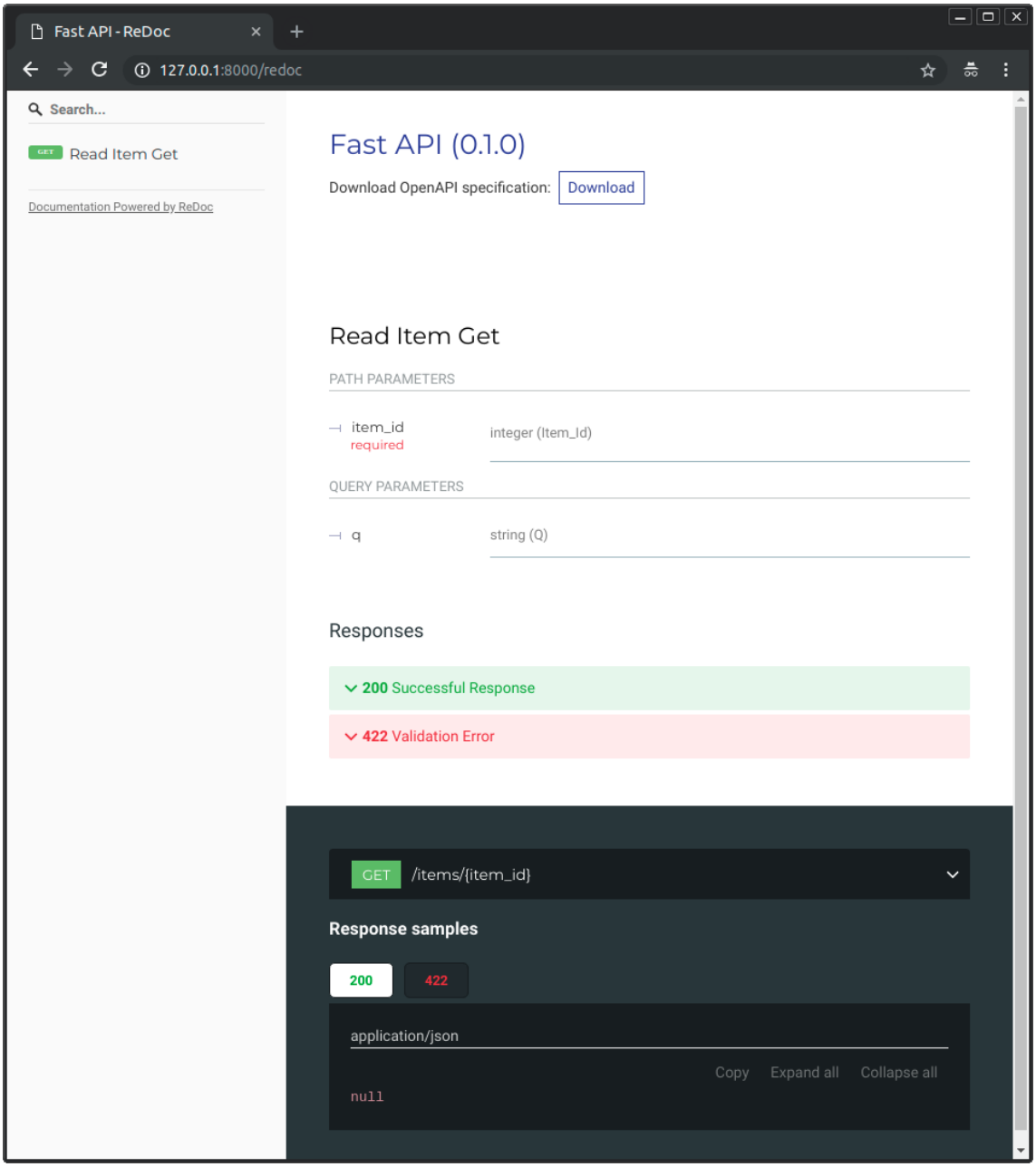
자동 대화형 API 문서를 볼 수 있습니다 (Swagger UI 제공):



대안 API 문서

그리고 이제 http://127.0.0.1:8000/redoc로 가봅시다.

다른 자동 문서를 볼 수 있습니다(ReDoc 제공):



예제 심화

이제 `PUT` 요청에 있는 본문(Body)을 받기 위해 `main.py` 를 수정해봅시다.

Pydantic을 이용해 파이썬 표준 타입으로 본문을 선언합니다.

```
from typing import Optional

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
```

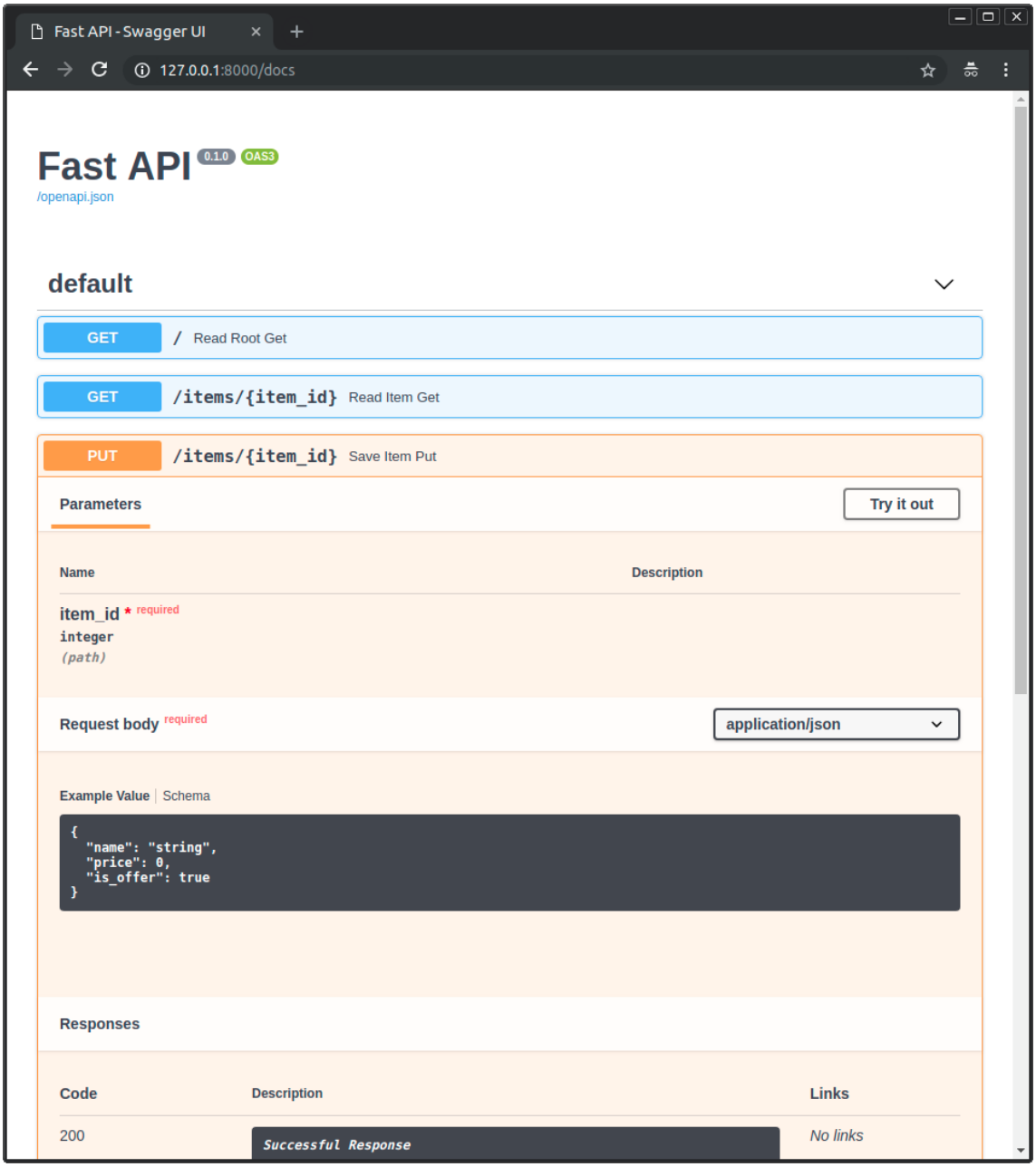
```python
class Item(BaseModel):
    name: str
    price: float
    is_offer: Optional[bool] = None


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


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[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}
```

서버가 자동으로 리로딩 할 수 있어야 합니다 (위에서 `uvicorn` 명령에 `--reload` 을 추가 했기 때문입니다).

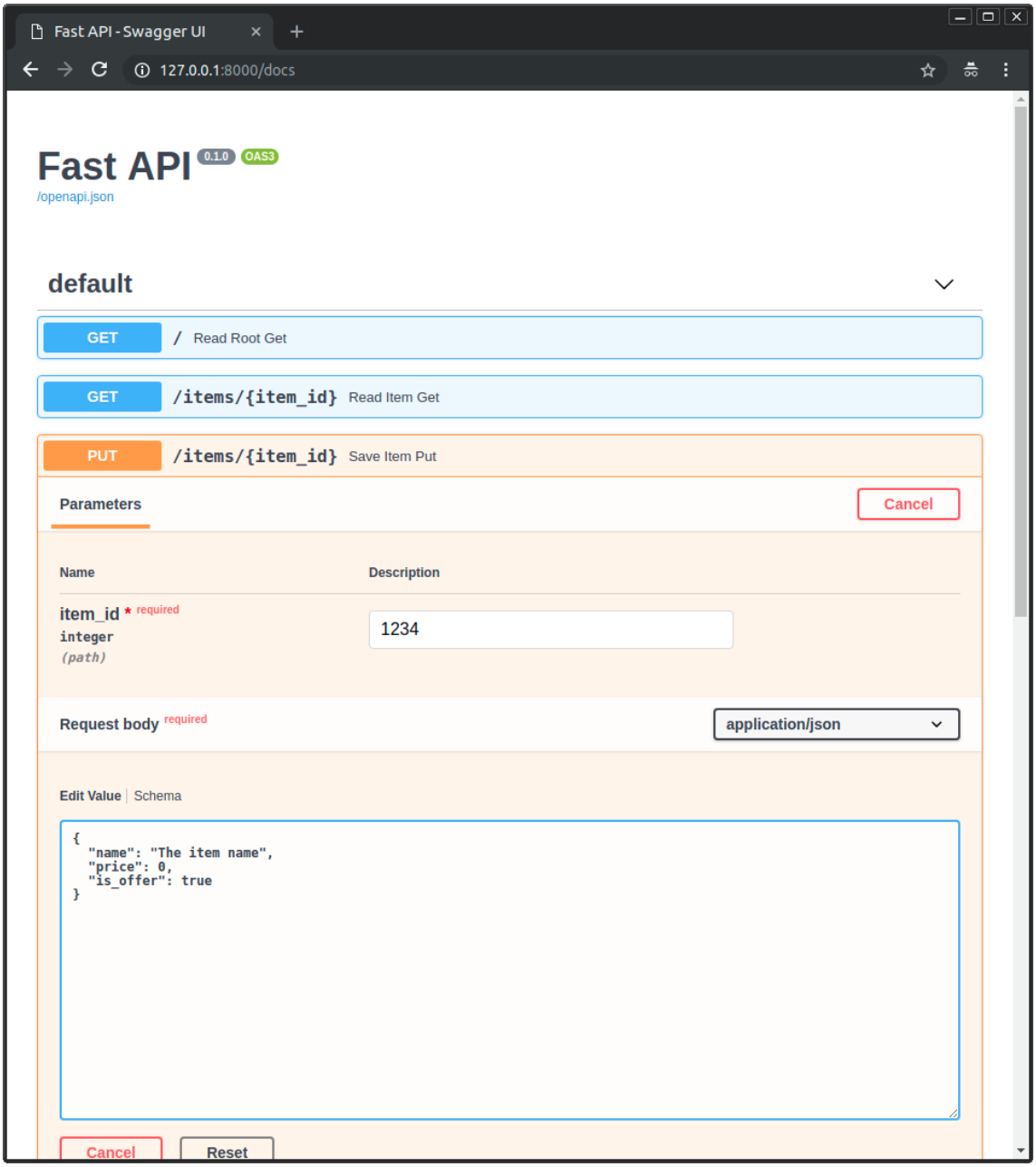
대화형 API 문서 업그레이드

이제 http://127.0.0.1:8000/docs로 이동합니다.

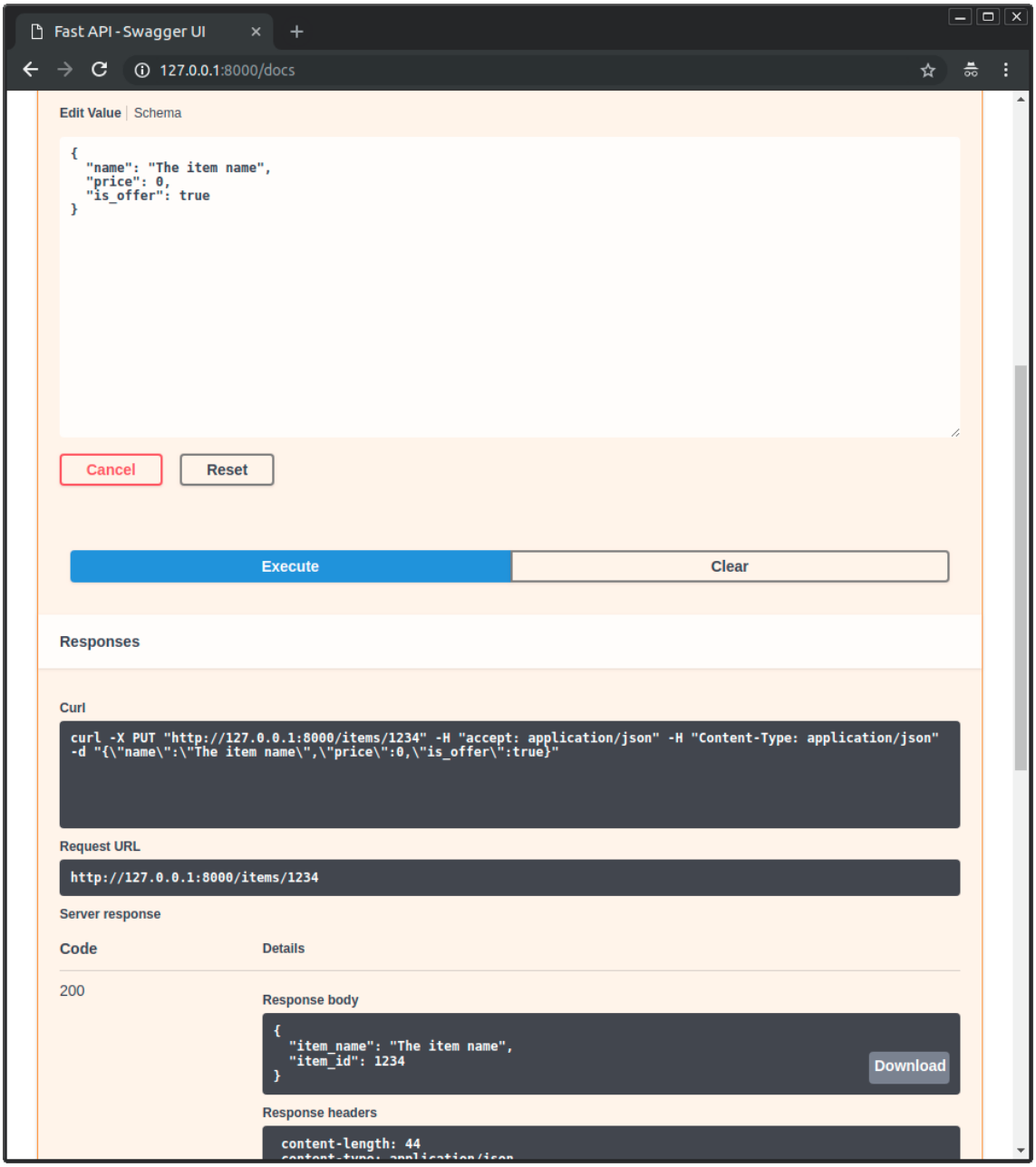
- 대화형 API 문서가 새 본문과 함께 자동으로 업데이트 합니다:



- "Try it out" 버튼을 클릭하면, 매개변수를 채울 수 있게 해주고 직접 API와 상호작용 할 수 있습니다:



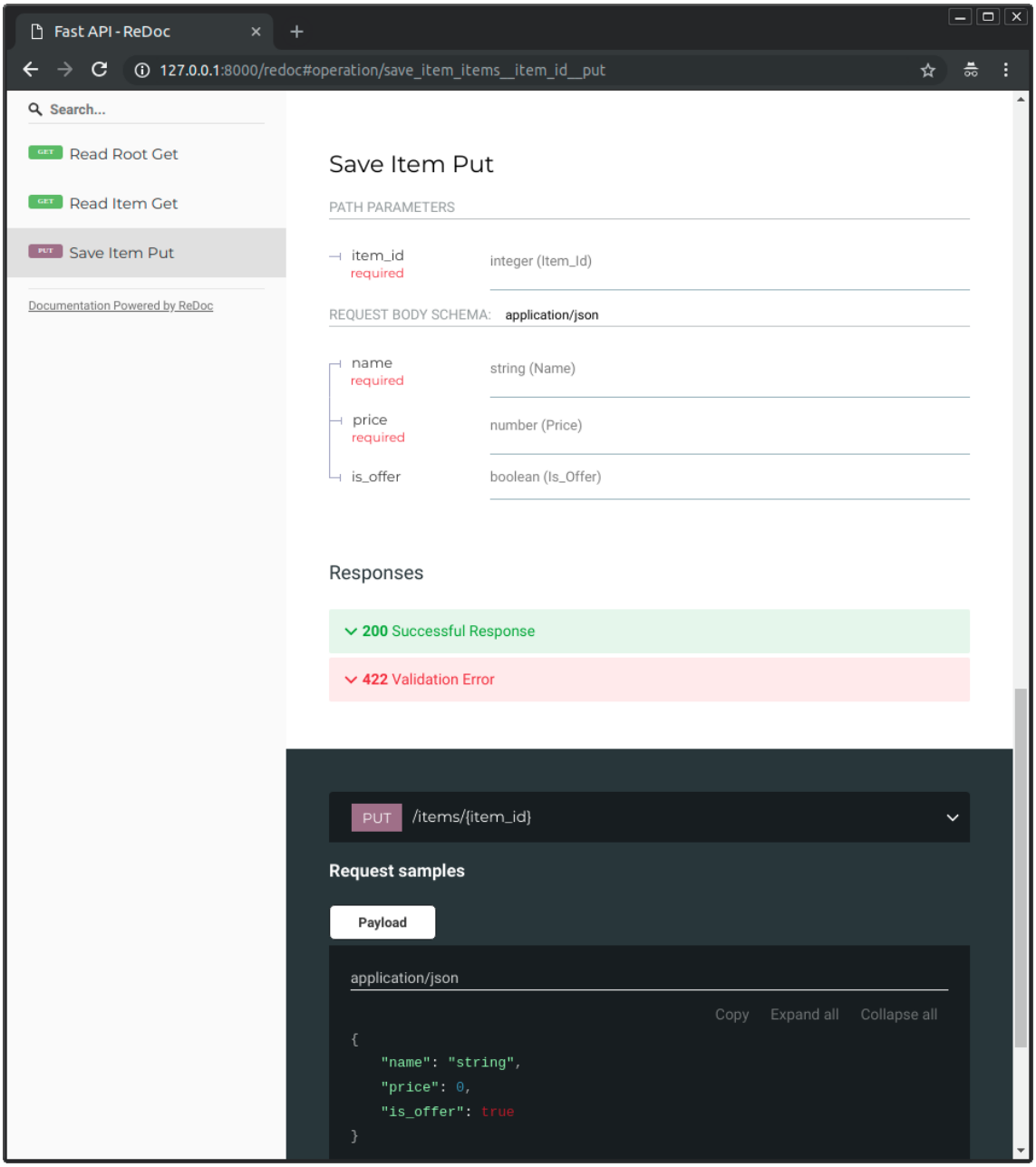
- 그러고 나서 "Execute" 버튼을 누르면, 사용자 인터페이스는 API와 통신하고 매개변수를 전송하며 그 결과를 가져와서 화면에 표시합니다:



대안 API 문서 업그레이드

그리고 이제, http://127.0.0.1:8000/redoc로 이동합니다.

- 대안 문서 역시 새 쿼리 매개변수와 본문을 반영합니다:



요약

요약하면, 여러분은 매개변수의 타입, 본문 등을 함수 매개변수로서 **한번에** 선언했습니다.

여러분은 현대 표준 파이썬 타입으로 이를 행했습니다.

새로운 문법, 특정 라이브러리의 메소드나 클래스 등을 배울 필요가 없습니다.

그저 표준 **Python 3.6+**입니다.

예를 들어, `int` 에 대해선:

```
item_id: int
```

또는 좀 더 복잡한 `Item` 모델에 대해선:

```
item: Item
```

...그리고 단 하나의 선언으로 여러분이 얻는 것은:

- 다음을 포함한 편집기 지원:
  - 자동완성.
  - 타입 검사.
- 데이터 검증:
  - 데이터가 유효하지 않을 때 자동으로 생성하는 명확한 에러.
  - 중첩된 JSON 객체에 대한 유효성 검사.
- 입력 데이터 변환: 네트워크에서 파이썬 데이터 및 타입으로 전송. 읽을 수 있는 것들:
  - JSON.
  - 경로 매개변수.
  - 쿼리 매개변수.
  - 쿠키.
  - 헤더.
  - 폼(Forms).
  - 파일.
- 출력 데이터 변환: 파이썬 데이터 및 타입을 네트워크 데이터로 전환(JSON 형식으로):
  - 파이썬 타입 변환 (`str` , `int` , `float` , `bool` , `list` , 등).
  - `datetime` 객체.
  - `UUID` 객체.
  - 데이터베이스 모델.
  - ...더 많은 것들.
- 대안가능한 사용자 인터페이스를 2개 포함한 자동 대화형 API 문서:
  - Swagger UI.
  - ReDoc.

이전 코드 예제로 돌아가서, **FastAPI**는 다음처럼 처리합니다:

- `GET` 및 `PUT` 요청에 `item_id` 가 경로에 있는지 검증.
- `GET` 및 `PUT` 요청에 `item_id` 가 `int` 타입인지 검증.
  - 그렇지 않다면 클라이언트는 유용하고 명확한 에러를 볼 수 있습니다.
- `GET` 요청에 `q` 라는 선택적인 쿼리 매개변수가 검사(`http://127.0.0.1:8000/items/foo?q=somequery` 처럼).
  - `q` 매개변수는 `= None` 으로 선언되었기 때문에 선택사항입니다.
  - `None` 이 없다면 필수사항입니다(`PUT` 의 경우와 마찬가지로).
- `/items/{item_id}` 으로의 `PUT` 요청은 본문을 JSON으로 읽음:
  - `name` 을 필수 속성으로 갖고 `str` 형인지 검사.
  - `price` 을 필수 속성으로 갖고 `float` 형인지 검사.
  - 만약 주어진다면, `is_offer` 를 선택 속성으로 갖고 `bool` 형인지 검사.
  - 이 모든 것은 깊이 중첩된 JSON 객체에도 적용됩니다.
- JSON을 변환하거나 JSON으로 변환하는 것을 자동화.
- 다음에서 사용할 수 있는 모든 것을 OpenAPI로 문서화:
  - 대화형 문서 시스템.

    - 여러 언어들에 대한 자동 클라이언트 코드 생성 시스템.
- 2개의 대화형 문서 웹 인터페이스를 직접 제공.

우리는 그저 수박 겉핥기만 했을 뿐인데 여러분은 벌써 어떻게 작동하는지 알고 있습니다.

다음 줄을 바꿔보십시오:

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

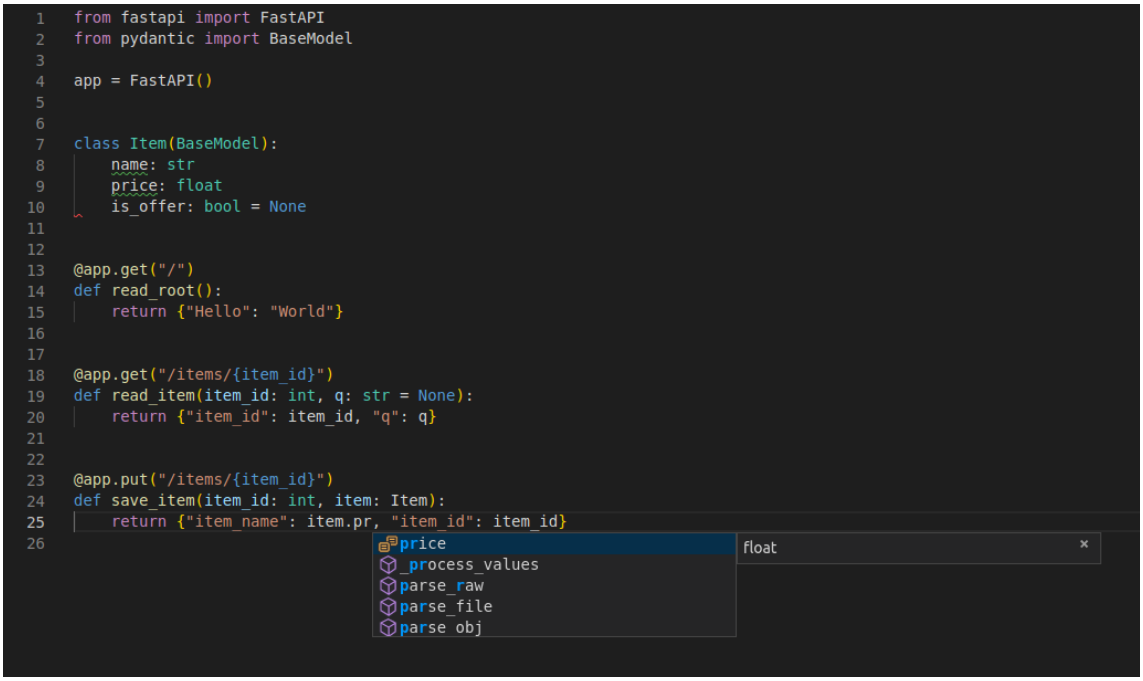
...에서:

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

...으로:

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

...그러고 나서 여러분의 편집기가 속성과 타입을 알고 자동 완성하는지 보십시오:



더 많은 기능을 포함한 보다 완전한 예제의 경우, 튜토리얼 - 사용자 가이드를 보십시오.

스포일러 주의: 튜토리얼 - 사용자 가이드는:

- 서로 다른 장소에서 **매개변수** 선언: **헤더**, **쿠키**, **폼 필드** 그리고 **파일**.
- `maximum_length` 또는 `regex` 처럼 **유효성 제약**하는 방법.
- 강력하고 사용하기 쉬운 **의존성 주입** 시스템.
- **OAuth2** 지원을 포함한 **JWT tokens** 및 **HTTP Basic**을 갖는 보안과 인증.
- (Pydantic 덕분에) **깊은 중첩 JSON 모델**을 선언하는데 더 진보한 (하지만 마찬가지로 쉬운) 기술.
- (Starlette 덕분에) 많은 추가 기능:

- **웹 소켓**
- **GraphQL**
- `requests` 및 `pytest` 에 기반한 극히 쉬운 테스트
- **CORS**
- **쿠키 세션**
- ...기타 등등.

성능

독립된 TechEmpower 벤치마크에서 Uvicorn에서 작동하는 FastAPI 어플리케이션이 사용 가능한 가장 빠른 프레임워크 중 하나로 Starlette와 Uvicorn(FastAPI에서 내부적으로 사용)에만 밑돌고 있습니다. (*)

자세한 내용은 벤치마크 섹션을 보십시오.

선택가능한 의존성

Pydantic이 사용하는:

- `ujson` - 더 빠른 JSON "파싱".
- `email_validator` - 이메일 유효성 검사.

Starlette이 사용하는:

- `requests` - `TestClient` 를 사용하려면 필요.
- `aiofiles` - `FileResponse` 또는 `StaticFiles` 를 사용하려면 필요.
- `jinja2` - 기본 템플릿 설정을 사용하려면 필요.
- `python-multipart` - `request.form()` 과 함께 "parsing"의 지원을 원하면 필요.
- `itsdangerous` - `SessionMiddleware` 지원을 위해 필요.
- `pyyaml` - Starlette의 `SchemaGenerator` 지원을 위해 필요 (FastAPI와 쓸때는 필요 없을 것입니다).
- `graphene` - `GraphQLApp` 지원을 위해 필요.
- `ujson` - `UJSONResponse` 를 사용하려면 필요.

FastAPI / Starlette이 사용하는:

- `uvicorn` - 애플리케이션을 로드하고 제공하는 서버.
- `orjson` - `ORJSONResponse` 을 사용하려면 필요.

`pip install fastapi[all]` 를 통해 이 모두를 설치 할 수 있습니다.

라이센스

이 프로젝트는 MIT 라이센스 조약에 따라 라이센스가 부여됩니다.