

{!./../docs/missing-translation.md!}



FastAPI framework, yüksek performanslı, öğrenmesi kolay, geliştirmesi hızlı, kullanıma sunulmaya hazır.

Test passing coverage 100% pypi package v0.81.0

dokümantasyon: <https://fastapi.tiangolo.com>

Kaynak kodu: <https://github.com/tiangolo/fastapi>



FastAPI, Python 3.6+'nın standart type hintlerine dayanan modern ve hızlı (yüksek performanslı) API'lar oluşturmak için kullanılabilecek web framework'ü.

Ana özellikleri:

- **Hızlı:** çok yüksek performanslı, **NodeJS** ve **Go** ile eşdeğer seviyede performans sağlıyor, (Starlette ve Pydantic sayesinde.) [Python'un en hızlı frameworklerinden bir tanesi.](#)
- **Kodlaması hızlı:** Yeni özellikler geliştirmek neredeyse %200 - %300 daha hızlı. *
- **Daha az bug:** Geliştirici (insan) kaynaklı hatalar neredeyse %40 azaltıldı. *
- **Sezgileri güçlü:** Editor (otomatik-tamamlama) desteği harika. [Otomatik tamamlama](#) her yerde. Debuglamak ile daha az zaman harcayacaksınız.
- **Kolay:** Öğrenmesi ve kullanması kolay olacak şekilde. Doküman okumak için harcayacağınız süre azaltıldı.
- **Kısa:** Kod tekrarını minimuma indirdik. Fonksiyon parametrelerinin tiplerini belirtmede farklı yollar sunarak karşılaşacağınız bug'ları azalttık.
- **Güçlü:** Otomatik dokümantasyon ile beraber, kullanıma hazır kod yaz.
- **Standartlar belirli:** Tamamiyle API'ların açık standartlara bağlı ve (tam uyumluluk içerisinde); [OpenAPI](#) (eski adıyla Swagger) ve [JSON Schema](#).

* Bahsi geçen rakamsal ifadeler tamamiyle, geliştirme takımının kendi sundukları ürünü geliştirirken yaptıkları testlere dayanmakta.

Sponsors

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

[Other sponsors](#)

Görüşler

"[...] Bugünlerde **FastAPI**'ı çok fazla kullanıyorum [...] Aslına bakarsanız **Microsoft**'taki **Machine Learning servislerimizin** hepsinde kullanmayı düşünüyorum. FastAPI ile geliştirdiğimiz servislerin bazıları çoktan **Windows**'un ana ürünlerine ve **Office** ürünlerine entegre edilmeye başlandı bile."

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

"**FastAPI**'ı **tahminlerimiz**'i sorgulanabilir hale getirmek için **REST** mimarısı ile beraber server üzerinde kullanmaya başladık."

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

"**Netflix kriz yönetiminde** orkestrasyon yapabilmek için geliştirdiği yeni framework'ü **Dispatch**'in, açık kaynak versiyonunu paylaşmaktan gurur duyuyor. [**FastAPI** ile yapıldı.]"

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

"**FastAPI** için ayın üzerindeymişçesine heyecanlıyım. Çok eğlenceli!"

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

"Dürüst olmak gerekirse, geliştirdiğin şey bir çok açıdan çok sağlam ve parlak görünüyor. Açıkcası benim **Hug**'ı tasarlariken yapmaya çalıştığım şey buydu - bunu birisinin başardığını görmek gerçekten çok ilham verici."

Timothy Crosley - **Hug**'ın Yaratıcısı [\(ref\)](#)

"Eğer REST API geliştirmek için **modern bir framework** öğrenme arayışında isen, **FastAPI**'a bir göz at [...] Hızlı, kullanımı ve öğrenmesi kolay. [...]"


"Biz **API** servislerimizi **FastAPI**'a geçirdik [...] Sizin de beğeneceğinizi düşünüyoruz. [...]"

Ines Montani - Matthew Honnibal - **Explosion AI** kurucuları - **spaCy** yaratıcıları [\(ref\)](#) - [\(ref\)](#)

Typer, komut satırı uygulamalarının FastAPI'ı



Eğer API yerine komut satırı uygulaması geliştiriyor isen **Typer**'a bir göz at.

Typer kısaca FastAPI'ın küçük kız kardeşi. Komut satırı uygulamalarının **FastAPI**'ı olması hedeflendi. 

Gereksinimler

Python 3.6+

FastAPI iki devin omuzları üstünde duruyor:

- Web tarafı için [Starlette](#).
- Data tarafı için [Pydantic](#).

Yükleme

```
$ pip install fastapi

---> 100%
```

Uygulamayı kullanılabilir hale getirmek için [Uvicorn](#) ya da [Hypercorn](#) gibi bir ASGI serverına ihtiyacın olacak.

```
$ pip install uvicorn[standard]

---> 100%
```

Örnek

Şimdi dene

- `main.py` adında bir dosya oluştur :

```
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}
```

► Ya da `async def...`

Çalıştır

Serverı aşağıdaki komut ile çalıştır:

```
$ 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.
```

► Çalıştırdığımız `uvicorn main:app --reload` hakkında...

Dokümantasyonu kontrol et

Browserını aç ve şu linke git <http://127.0.0.1:8000/items/5?q=somequery>.

Bir JSON yanıtı göreceksin:

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

Az önce oluşturduğun API:

- `/` ve `/items/{item_id}` adreslerine HTTP talebi alabilir hale geldi.
- İki *adresde* `GET` *operasyonlarını* (HTTP *metodları* olarak bilinen) yapabilir hale geldi.
- `/items/{item_id}` *adres*i ayrıca bir `item_id` *adres parametresine* sahip ve bu bir `int` olmak zorunda.
- `/items/{item_id}` *adres*i opsiyonel bir `str` *sorgu paramtersine* sahip bu da `q`.

İnteraktif API dokümantasyonu

Şimdi <http://127.0.0.1:8000/docs> adresine git.

Senin için otomatik oluşturulmuş([Swagger UI](#) tarafından sağlanan) interaktif bir API dokümanı göreceksin:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Fast API 0.1.0 OAS3

/openapi.json

default

GET /items/{item_id} Read Item Get

Try it out

Parameters

Name	Description
item_id * required	
integer	
(path)	
q	
string	
(query)	

Responses

Code	Description	Links
200	Successful Response	No links
	application/json	
	Controls Accept header.	
422	Validation Error	No links
	application/json	

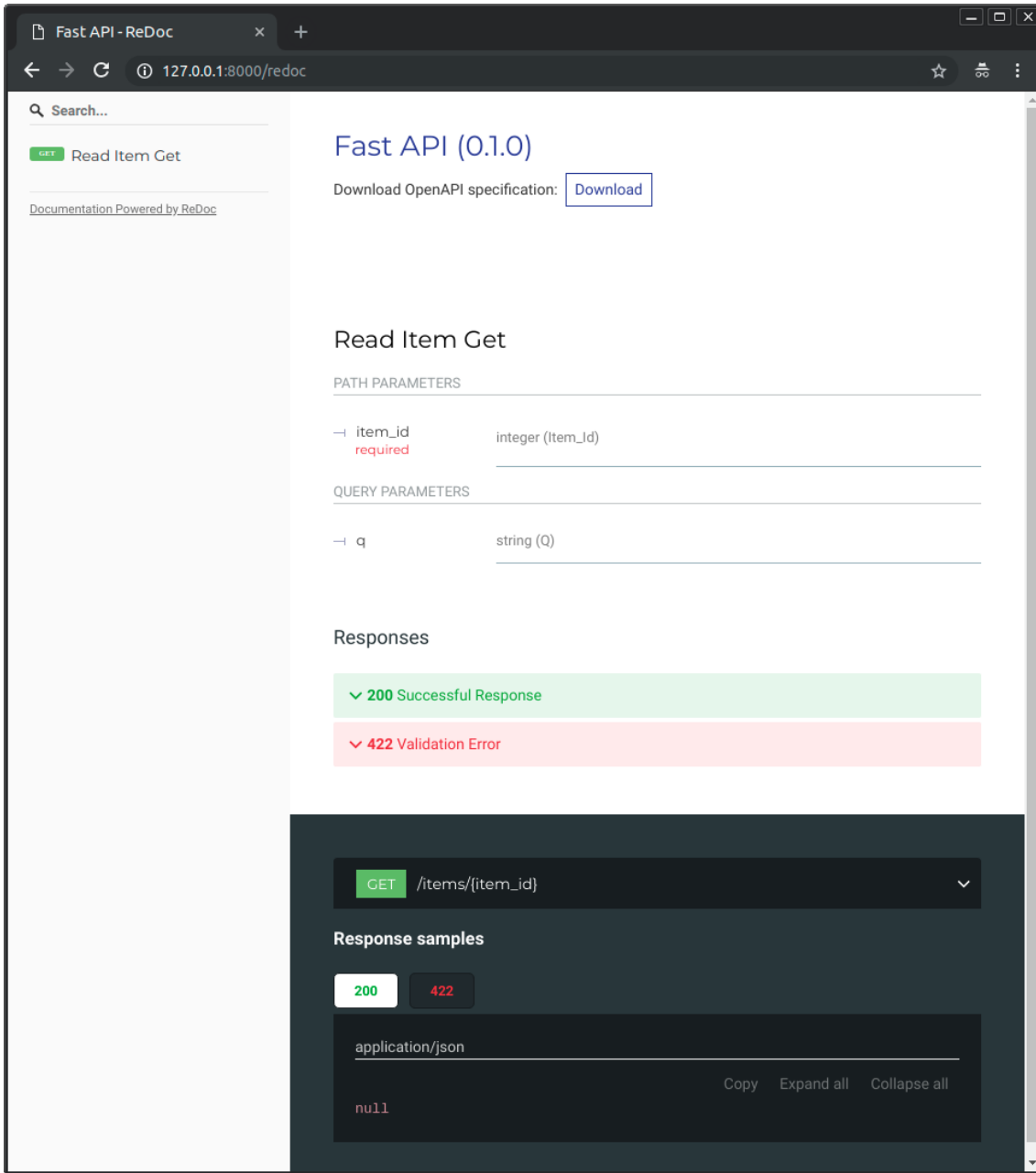
Example Value | Schema

```
{  "detail": [    {      "loc": [        "string"      ]    }  ]}
```

Alternatif API dokümantasyonu

Şimdi <http://127.0.0.1:8000/redoc> adresine git.

Senin için alternatif olarak ([ReDoc](#) tarafından sağlanan) bir API dokümantasyonu daha göreceksin:



Örnek bir değişiklik

Şimdi `main.py` dosyasını değiştirelim ve body ile `PUT` talebi alabilir hale getirelim.

Şimdi Pydantic sayesinde, Python'un standart tiplerini kullanarak bir body tanımlayacağız.

```
from typing import Optional

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
```

```
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}
```

Server otomatik olarak yeniden başlamalı (çünkü yukarıda `uvicorn` 'u çalıştırırken `--reload` parametresini kullandık.).

İnteraktif API dokümantasyonu'nda değiştirme yapmak

Şimdi <http://127.0.0.1:8000/docs> bağlantısına tekrar git.

- İnteraktif API dokümantasyonu, yeni body ile beraber çoktan yenilenmiş olması lazım:

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"a tıkla, bu senin API parametleri üzerinde deneme yapabilmene izin veriyor:

Fast API - Swagger UI

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

Cancel

Name	Description
item_id * required integer (path)	1234

Request body required

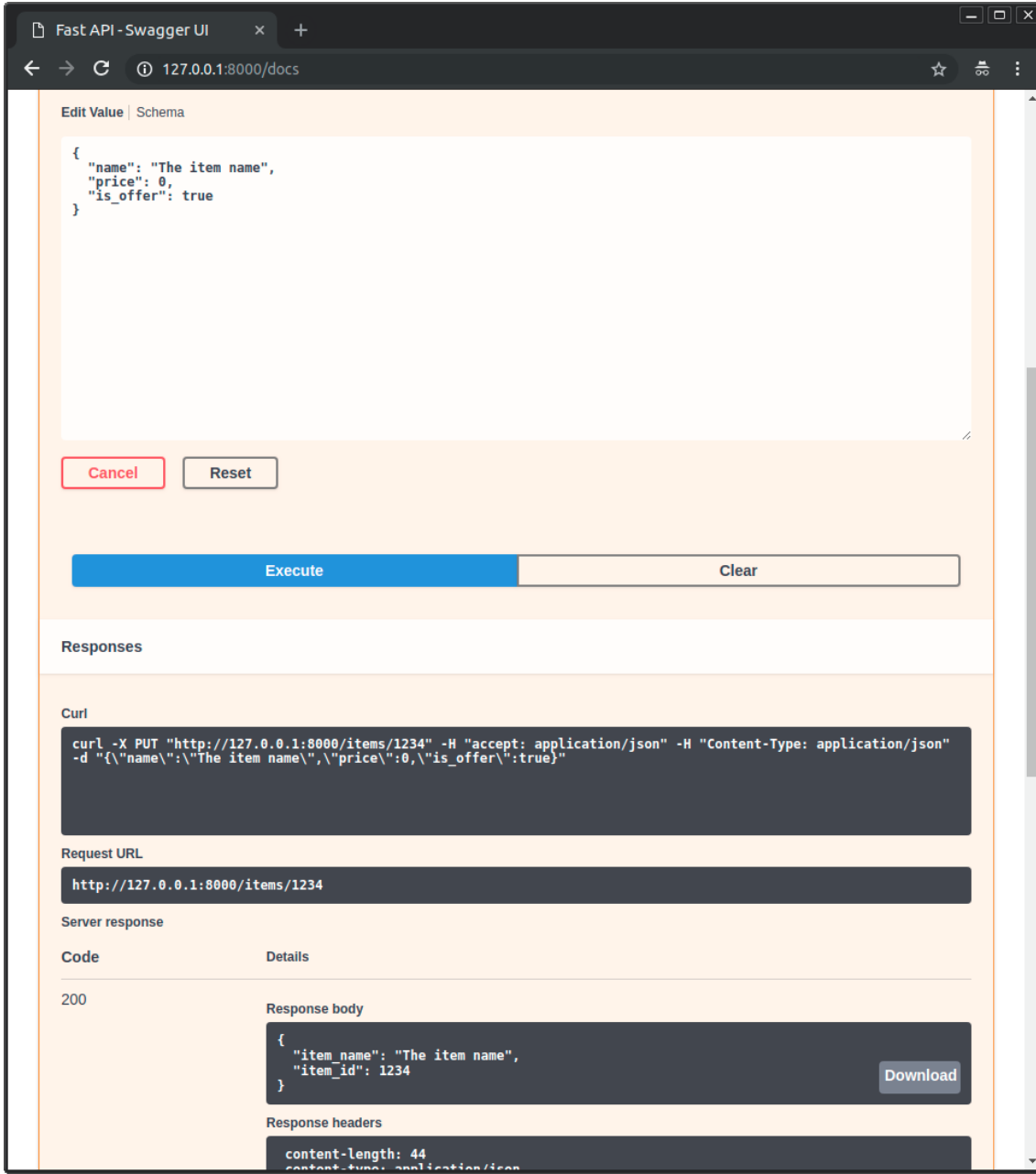
application/json

Edit Value | Schema

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

Cancel Reset

- Şimdi "Execute" butonuna tıkla, kullanıcı arayüzü otomatik olarak API'ın ile bağlantı kurarak ona bu parametreleri gönderecek ve sonucu karşına getirecek.



Alternatif API dokümantasyonunda değiřtirmek

řimdi ise <http://127.0.0.1:8000/redoc> adresine git.

- Alternatif dokümantasyonda koddaki değışimlerle beraber kendini yeni query ve body ile güncelledi.

Fast API - ReDoc

127.0.0.1:8000/redoc#operation/save_item_items__item_id__put

Search...

GET Read Root Get

GET Read Item Get

PUT Save Item Put

Documentation Powered by ReDoc

Save Item Put

PATH PARAMETERS

item_id	integer (Item_Id)
required	

REQUEST BODY SCHEMA: application/json

name	string (Name)
required	
price	number (Price)
required	
is_offer	boolean (Is_Offer)

Responses

- ✓ 200 Successful Response
- ✓ 422 Validation Error

PUT /items/{item_id}

Request samples

Payload

application/json

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

Copy Expand all Collapse all

Özet

Özetleyecek olursak, URL, sorgu veya request body'deki parametrelerini fonksiyon parametresi olarak kullanıyorsun. Bu parametrelerin veri tiplerini bir kere belirtmen yeterli.

Type-hinting işlemini Python dilindeki standart veri tipleri ile yapabilirsin

Yeni bir syntax'e alışmana gerek yok, metodlar ve classlar zaten spesifik kütüphanelere ait.

Sadece standart **Python 3.6+**.

Örnek olarak, `int` tanımlamak için:

```
item_id: int
```

ya da daha kompleks `Item` tipi:

```
item: Item
```

...sadece kısa bir parametre tipi belirtmekle beraber, sahip olacakların:

- Editör desteği dahil olmak üzere:
 - Otomatik tamamlama.
 - Tip sorguları.
- Datanın tipe uyumunun sorgulanması:
 - Eğer data geçersiz ise, otomatik olarak hataları ayıklar.
 - Çok derin JSON objelerinde bile veri tipi sorgusu yapar.
- Gelen verinin dönüşümünü aşağıdaki veri tiplerini kullanarak gerçekleştirebiliyor.
 - JSON.
 - Path parametreleri.
 - Query parametreleri.
 - Cookies.
 - Headers.
 - Forms.
 - Files.
- Giden verinin dönüşümünü aşağıdaki veri tiplerini kullanarak gerçekleştirebiliyor (JSON olarak):
 - Python tiplerinin (`str` , `int` , `float` , `bool` , `list` , vs) çevirisi.
 - `datetime` objesi.
 - `UUID` objesi.
 - Veritabanı modelleri.
 - ve daha fazlası...
- 2 alternatif kullanıcı arayüzü dahil olmak üzere, otomatik interaktif API dokümanı:
 - Swagger UI.
 - ReDoc.

Az önceki kod örneğine geri dönelim, **FastAPI**'in yapacaklarına bir bakış atalım:

- `item_id` 'nin `GET` ve `PUT` talepleri içinde olup olmadığının doğruluğunu kontrol edecek.
- `item_id` 'nin tipinin `int` olduğunu `GET` ve `PUT` talepleri içinde olup olmadığının doğruluğunu kontrol edecek.
 - Eğer `GET` ve `PUT` içinde yok ise ve `int` değil ise, sebebini belirten bir hata mesajı gösterecek
- Opsiyonel bir `q` parametresinin `GET` talebi için (`http://127.0.0.1:8000/items/foo?q=somequery` içinde) olup olmadığını kontrol edecek
 - `q` parametresini `= None` ile oluşturduğumuz için, opsiyonel bir parametre olacak.
 - Eğer `None` olmasa zorunlu bir parametre olacak idi (bu yüzden body'de `PUT` parametresi var).
- `PUT` talebi için `/items/{item_id}` 'nin body'sini, JSON olarak okuyor:
 - `name` adında bir parametreye olup olmadığını ve var ise onun `str` olup olmadığını kontrol ediyor.

- `price` adında bir parametreye olup olmadığını ve var ise onun `float` olup olmadığını kontrol ediyor.
- `is_offer` adında bir parametreye olup olmadığını ve var ise onun `bool` olup olmadığını kontrol ediyor.
- Bunların hepsini en derin JSON modellerinde bile yapacaktır.
- Bütün veri tiplerini otomatik olarak JSON'a çeviriyor veya tam tersi.
- Her şeyi dokümanlayıp, çeşitli yerlerde:
 - İnteraktif dokümantasyon sistemleri.
 - Otomatik alıcı kodu üretim sistemlerinde ve çeşitli dillerde.
- İki ayrı web arayüzüyle direkt olarak interaktif bir dokümantasyon sunuyor.

Henüz yüzeysel bir bakış attık, fakat sen çoktan çalışma mantığını anladın.

Şimdi aşağıdaki satırı değiştirmeyi dene:

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

...bundan:

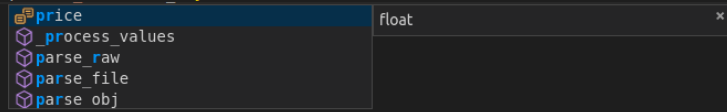
```
... "item_name": item.name ...
```

...buna:

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

...şimdi editör desteğinin nasıl veri tiplerini bildiğini ve otomatik tamamladığını gör:

```
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
```



Daha fazla örnek ve özellik için [Tutorial - User Guide](#) sayfasını git.

Spoiler: Öğretici - Kullanıcı rehberi şunları içeriyor:

- **Parameterlerini** nasıl **headers**, **cookies**, **form fields** ve **files** olarak deklare edebileceğini.
- `maximum_length` ya da `regex` gibi şeylerle nasıl **doğrulama** yapabileceğini.
- Çok güçlü ve kullanımı kolay **Zorunluluk Entegrasyonu** oluşturmayı.
- Güvenlik ve kimlik doğrulama, **JWT tokenleri**'yle beraber **OAuth2** desteği, ve **HTTP Basic** doğrulaması.
- İleri seviye fakat ona göre oldukça basit olan **derince oluşturulmuş JSON modelleri** (Pydantic sayesinde).
- Diğer ekstra özellikler (Starlette sayesinde):
 - **WebSockets**
 - **GraphQL**
 - `requests` ve `pytest` sayesinde aşırı kolay testler.
 - **CORS**
 - **Cookie Sessions**
 - ...ve daha fazlası.

Performans

Bağımsız TechEmpower kıyaslamaları gösteriyor ki, Uvicorn'la beraber çalışan **FastAPI** uygulamaları [Python'un en hızlı frameworklerinden birisi](#), sadece Starlette ve Uvicorn'dan daha yavaş ki FastAPI bunların üzerine kurulu.

Daha fazla bilgi için, bu bölüme bir göz at [Benchmarks](#).

Opsiyonel gereksinimler

Pydantic tarafında kullanılan:

- [ujson](#) - daha hızlı JSON "dönüşümü" için.
- [email_validator](#) - email doğrulaması için.

Starlette tarafında kullanılan:

- [requests](#) - Eğer `TestClient` kullanmak istiyorsan gerekli.
- [aiofiles](#) - `FileResponse` ya da `StaticFiles` kullanmak istiyorsan gerekli.
- [jinja2](#) - Eğer kendine ait template konfigürasyonu oluşturmak istiyorsan gerekli
- [python-multipart](#) - Form kullanmak istiyorsan gerekli ("dönüşümü").
- [itsdangerous](#) - `SessionMiddleware` desteği için gerekli.
- [pyyaml](#) - `SchemaGenerator` desteği için gerekli (Muhtemelen FastAPI kullanırken ihtiyacınız olmaz).
- [graphene](#) - `GraphQLApp` desteği için gerekli.
- [ujson](#) - `UJSONResponse` kullanmak istiyorsan gerekli.

Hem FastAPI hem de Starlette tarafından kullanılan:

- [uvicorn](#) - oluşturduğumuz uygulamayı bir web sunucusuna servis etmek için gerekli
- [orjson](#) - `ORJSONResponse` kullanmak istiyor isen gerekli.

Bunların hepsini `pip install fastapi[all]` ile yükleyebilirsiniz.

Lisans

Bu proje, MIT lisansı şartlarına göre lisanslanmıştır.