

Özellikler

FastAPI özellikleri

FastAPI sana bunları sağlıyor

Açık standartları temel alır

- API oluşturma işlemlerinde [OpenAPI](#) buna path operasyonları parametreleri, body talebi, güvenlik gibi şeyler dahil olmak üzere deklare bunların deklare edilmesi.
- Otomatik olarak data modelinin [JSON Schema](#) ile beraber dokümente edilmesi (OpenAPI'n kendisi zaten JSON Schema'ya dayanıyor).
- Titiz bir çalışmanın sonucunda yukarıdaki standartlara uygun bir framework oluşturduk. Standartları pastanın üzerine sonradan eklenmiş bir çilek olarak görmedik.
- Ayrıca bu bir çok dilde kullanılabiləcək **client code generator** kullanımına da izin veriyor.

Otomatik dokümantasyon

OpenAPI standartlarına dayalı olan bir framework olarak, geliştiricilerin birden çok seçeneği var, varsayılan olarak gelen 2 farklı interaktif API dokümantasyonu ve web kullanıcı arayüzü var.

- [Swagger UI](#) interaktif olarak API'nızı tarayıcı üzerinden çağırıp test edebilmenize olanak sağlıyor.

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

- [ReDoc](#) ile beraber alternatif API dokümantasyonu.

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)
---------	-------------------

REQUEST BODY SCHEMA: application/json

name	string (Name)
price	number (Price)
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

Sadece modern Python

Tamamiyle standartlar **Python 3.6**'nın type hintlerine dayanıyor (Pydantic'in sayesinde). Yeni bir syntax öğrenmene gerek yok. Sadece modern Python.

Eğer Python type hintlerini bilmiyorsan veya bir hatırlatmaya ihtiyacın var ise(FastAPI kullanmasan bile) şu iki dakikalık küçük bilgilendirici içeriğe bir göz at: [Python Types](#){internal-link target=_blank}.

Standart Python'u typelarını belirterek yazıyorsun:

```
from typing import List, Dict
from datetime import date
```

```
from pydantic import BaseModel

# Değişkeni str olarak belirt
# ve o fonksiyon için harika bir editör desteği al
def main(user_id: str):
    return user_id

# Pydantic modeli
class User(BaseModel):
    id: int
    name: str
    joined: date
```

Sonrasında bu şekilde kullanabilirsin

```
my_user: User = User(id=3, name="John Doe", joined="2018-07-19")

second_user_data = {
    "id": 4,
    "name": "Mary",
    "joined": "2018-11-30",
}

my_second_user: User = User(**second_user_data)
```

!!! info `**second_user_data` şu anlama geliyor:

```
Key-Value çiftini direkt olarak `second_user_data` dictionarysine kaydet , yaptığın
şey buna eşit olacak: `User(id=4, name="Mary", joined="2018-11-30")`
```

Editor desteği

Bütün framework kullanılması kolay ve sezgileri güçlü olması için tasarlandı, verilen bütün kararlar geliştiricilere en iyi geliştirme deneyimini yaşatmak üzere, bir çok editör üzerinde test edildi.

Son yapılan Python geliştiricileri anketinde, açık ara [en çok kullanılan özellik "oto-tamamlama" idi.](#)

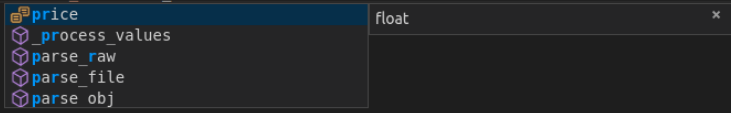
Bütün **FastAPI** frameworkü oto-tamamlama açısından geliştiriciyi tatmin etmek üzerine tasarlandı. Otomatik tamamlama her yerde çalışıyor.

Dokümantasyona tekrardan çok nadir olarak geleceksin.

Editörün sana nasıl yardım ettiğine bir bak:

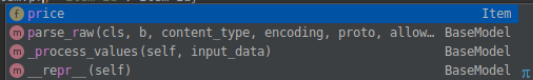
- [Visual Studio Code](#) ile:

```
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.name, "item_id": item_id}
26
```



- [PyCharm](#) ile:

```
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.name, "item_id": item_id}
26
```



Daha önceden düşünüp en imkansız diyebileceğin durumlarda bile otomatik tamamlama alacaksın, örnek olarak `price` JSON body içerisinde (nested bir JSON body de olabilirdi.) direkt olarak istekten geliyor, bu durumda bile oto-tamamlama sağlıyor.

Artık key isimlerini yanlış yazma, dokümantasyona dönüp deliller gibi yukarı aşağı sayfada gezmek ve en sonunda `username` mi yoksa `user_name` mi kullandım gibi sorular yok.

Kısa

Her şey için mantıklı bir **varsayılanı** var. Parametrelerini opsiyonel olarak tanımlayıp API'nı istediğin gibi modifiye edebilirsin.

Hepsi varsayılan olarak **çalışıyor**.

Doğrulama

- Neredeyse bütün (ya da hepsi?) Python **data tipleri** için doğrulama, kapsadıkları:
 - JSON objeleri (`dict`).
 - JSON array (`list`) item type'ı belirtirken.
 - String (`str`) parametresi, minimum ve maksimum uzunluk gibi sınırlandırmalar yaparken.
 - Numaralar (`int` , `float`) maksimum ve minimum gibi sınırlandırmalar yaparken.
- Bunlar gibi en egzotik tiplerle bile doğrulama yapabiliyorsunuz.:
 - URL.
 - Email.
 - UUID.
 - ...ve diğerleri.

Bütün doğrulama olayları çok güçlü bir kütüphane sayesinde yapılıyor, **Pydantic**.

Güvenlik ve kimlik doğrulama

Güvenlik ve doğrulama database ve data modellerinden taviz vermeden entegre edilebilir durumda.

Bütün güvenlik şemaları OpenAPI'da tanımlanmış durumda, kapsadıkları:

- HTTP Basic.
- **OAuth2** (ve **JWT tokenleriyle** beraber). Bu öğretici içeriğe göz atabilirsin [OAuth2 with JWT](#){internal-link target=_blank}.
- API anahtarları:
 - Headerlar.
 - Query parametreleri.
 - Cookies, vs.

Bütün güvenlik özellikleri Starlette'den geliyor (**session cookies'de** dahil olmak üzere).

Bütün hepsi tekrardan kullanılabilir aletler ve bileşenler olarak, kolayca sistemlerinize, data depolarınıza, ilişkisel ve NoSQL databaseslerinize entegre edebileceğiniz şekilde yapıldı.

Dependency injection

FastAPI'ın inanılmaz derecede kullanımı kolay, fakat inanılmaz derecede güçlü **Dependency Injection** sistemi var.

- Dependencylerin bile dependencies'i olabiliyor, FastAPI bunun için **graph of "dependency"** yaratıyor.
- Hepsi **otomatik olarak** FastAPI tarafından hallediliyor.
- Bütün zorunlulukların gelen datalara bağlı olarak farklı gereksinimleri olabiliyor, ilave path operasyonlarının kısıtlamaları ve otomatik dokümantasyonu da ayrıca yapılıyor .
- Path operasyonu parametreleri içerisinde belirtilen gereksinimler için bile **Otomatik doğrulama** yapılabilir.
- Kompleks kimlik doğrulama sistemleri için destek, **database bağlantıları**, vs.
- **Taviz yok** hiçbir şeyden taviz vermeden, database frontend vs. Bütün hepsinin kolayca entegre edilebilir.

Sınırsız "plug-inler"

Başka bir deyişle, plug-inlere ihtiyacımız yok, import edip direkt olarak kullanmaya başlayabiliriz.

Bütün entegrasyonlar kullanımı kolay olmak üzere (zorunluluklar ile beraber) tasarlandı, sen bir "plug-in" yaratıp 2 satır kod ile, *path operasyonlarında* kullandığımız syntax ve aynı yapı ile koduna entegre edebilirsiniz.

Test edildi

- 100% [test coverage](#).
- 100% [typeları belirtilmiş](#) codebase.
- FastAPI ile yapılan bir çok proje insanlar tarafından kullanılıyor.

Starlette özellikleri

FastAPI, [Starlette](#) ile tamamiyle uyumlu ve üzerine kurulu. Yani FastAPI üzerine ekleme yapacağınız herhangi bir Starlette kodu da çalışacaktır.

`FastAPI` aslında `Starlette` 'nin bir sub-class'ı. Eğer Starlette'nin nasıl kullanılacağını biliyor isen, çoğu işlevini aynı şekilde yapıyor.

FastAPI ile beraber **Starlette**'nin bütün özelliklerine de sahip olacaksınız (FastAPI aslında Starlette'nin steroid basmış hali):

- Gerçekten etkileyici bir performansa sahip.Python'un ise en hızlı frameworklerinden bir tanesi, [NodeJS ve Go ile ise eşdeğer performansa sahip.](#)
- **WebSocket** desteği.
- **GraphQL** desteği.
- Kullanım halinde arka plan işlevleri.
- Başlatma ve kapatma eventleri(startup and shutdown).
- Test sunucusu `requests` üzerine kurulu.
- **CORS**, GZip, Static dosyalar, Streaming responseları.
- **Session and Cookie** desteği.
- 100% test kapsayıcılığı.
- 100% [typeları belirtilmiş](#) codebase.

Pydantic özellikleri

FastAPI ile [Pydantic](#) tamamiyle uyumlu ve üzerine kurulu. Yani FastAPI üzerine ekleme yapacağınız herhangi bir Pydantic kodu da çalışacaktır.

Bunlara Pydantic üzerine kurulu [ORM](#) databaseler ve , [ODM](#) kütüphaneler de dahil olmak üzere.

Bu ayrıca şu anlama da geliyor, bir çok durumda requestten gelen objeyi **direkt olarak database'e** her şeyi otomatik olarak doğrulanmış bir biçimde aktarabilirsiniz.

Aynı şekilde, databaseden gelen objeyi de **direkt olarak isteğe** de tamamiyle doğrulanmış bir biçimde gönderebilirsiniz.

FastAPI ile beraber **Pydantic**'in bütün özelliklerine sahip olacaksınız (FastAPI data kontrolünü Pydantic'in üzerine kurduğu için):

- **Kafa karıştırmaz:**
 - Farklı bir syntax öğrenmenize gerek kalmaz,
 - Eğer Python typelarını nasıl kullanacağını biliyorsan Pydantic kullanmayı da biliyorsunuzdur.
- Kullandığın geliştirme araçları ile iyi çalışır [IDE/linter/brain](#):
 - Pydantic'in veri yapıları aslında sadece senin tanımladığın classlar; Bu yüzden doğrulanmış dataların ile otomatik tamamlama, linting ve mypy'ı kullanarak sorunsuz bir şekilde çalışabilirsin
- **Hızlı:**
 - [Benchmarklarda](#), Pydantic'in diğer bütün test edilmiş bütün kütüphanelerden daha hızlı.

- **En kompleks** yapıları bile doğrula:
 - Hiyerarşik Pydantic modellerinin kullanımı ile beraber, Python `typing`'s `List` and `Dict` , vs gibi şeyleri doğrula.
 - Doğrulayıcılar en kompleks data şemalarının bile temiz ve kolay bir şekilde tanımlanmasına izin veriyor, ve hepsi JSON şeması olarak dokümente ediliyor
 - Pydantic, JSON objen ne kadar derin (nested) olursa olsun doğrulamasını ve gösterimini yapıyor
- **Geniştirilebilir:**
 - Pydantic özelleştirilmiş data tiplerinin tanımlanmasının yapılmasına izin veriyor ayrıca validator decoratorü ile senin doğrulamaları genişletip, kendi doğrulayıcılarını yazmana izin veriyor.
- 100% test kapsayıcılığı.