Intro to FastAPI

High performance, easy to learn,
fast to code, ready for production

Who am I?

Sebastián Ramírez

tiangolo.com

Dev at Explosion

Berlin, Germany

github.com/tiangolo

linkedin.com/in/tiangolo

twitter.com/tiangolo

Explosion created:

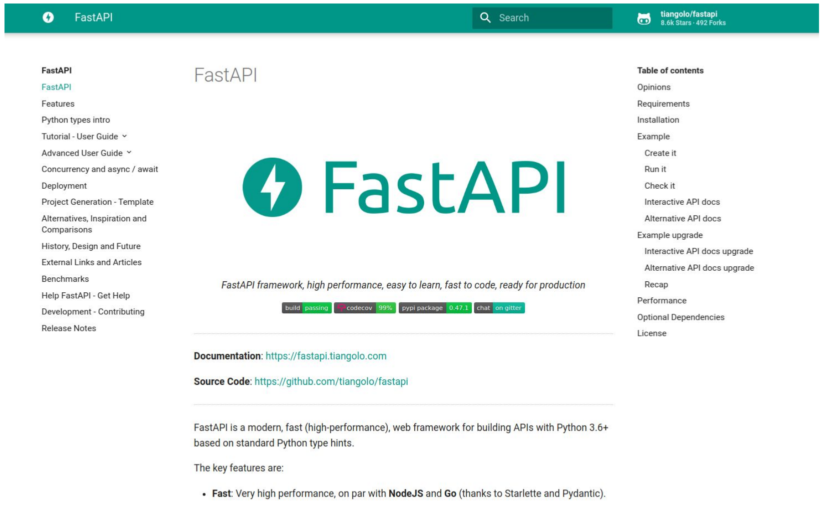
spaCy prodigy THiNC

I created:

FastAPI Typer

About **FastAPI**

- 19K GitHub stars (about 1K+ per month)
- Used by Microsoft, Uber, Netflix, etc.
- Performance in the top rank for Python


FastAPI

Search

tiangolo/fastapi
8.6k Stars · 492 Forks

FastAPI
FastAPI
Features
Python types intro
Tutorial - User Guide
Advanced User Guide
Concurrency and async / await
Deployment
Project Generation - Template
Alternatives, Inspiration and Comparisons
History, Design and Future
External Links and Articles
Benchmarks
Help FastAPI - Get Help
Development - Contributing
Release Notes

FastAPI

FastAPI

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

build passing · codecov 99% · pypi package 0.47.1 · chat on gitter

Documentation: https://fastapi.tiangolo.com

Source Code: https://github.com/tiangolo/fastapi

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

The key features are:

- **Fast**: Very high performance, on par with **NodeJS** and **Go** (thanks to Starlette and Pydantic).

Table of contents

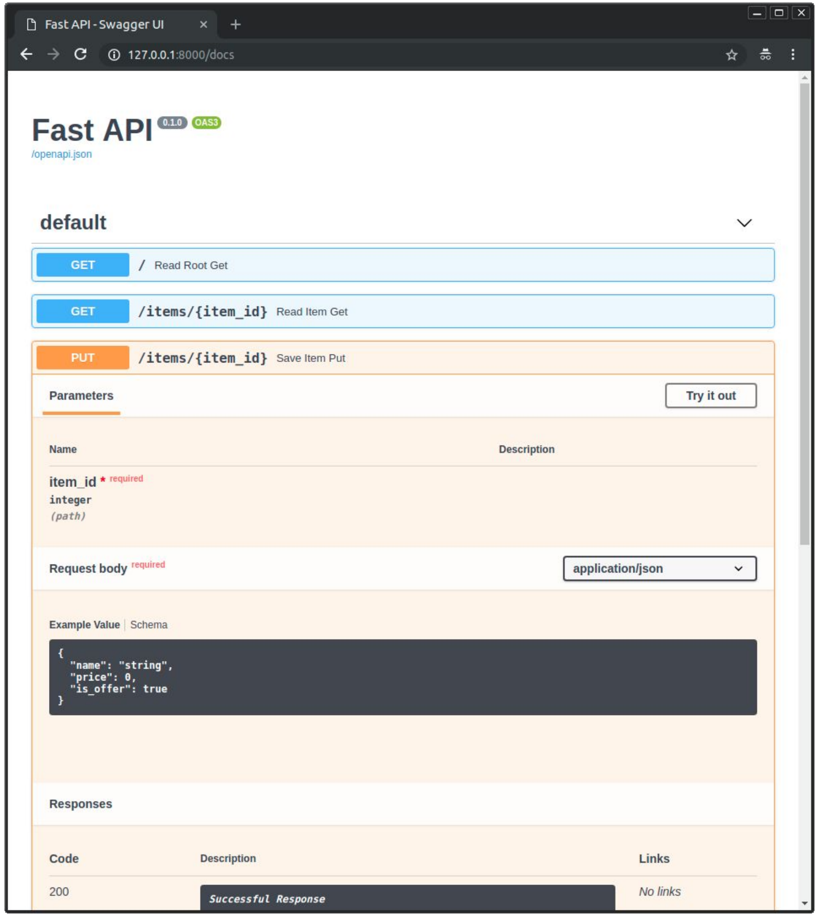
Opinions
Requirements
Installation
Example
  Create it
  Run it
  Check it
  Interactive API docs
  Alternative API docs
Example upgrade
  Interactive API docs upgrade
  Alternative API docs upgrade
  Recap
Performance
Optional Dependencies
License

@tiangolo

Based on standards

- OpenAPI
- JSON Schema
- OAuth2

✨ Automatic API docs ✨



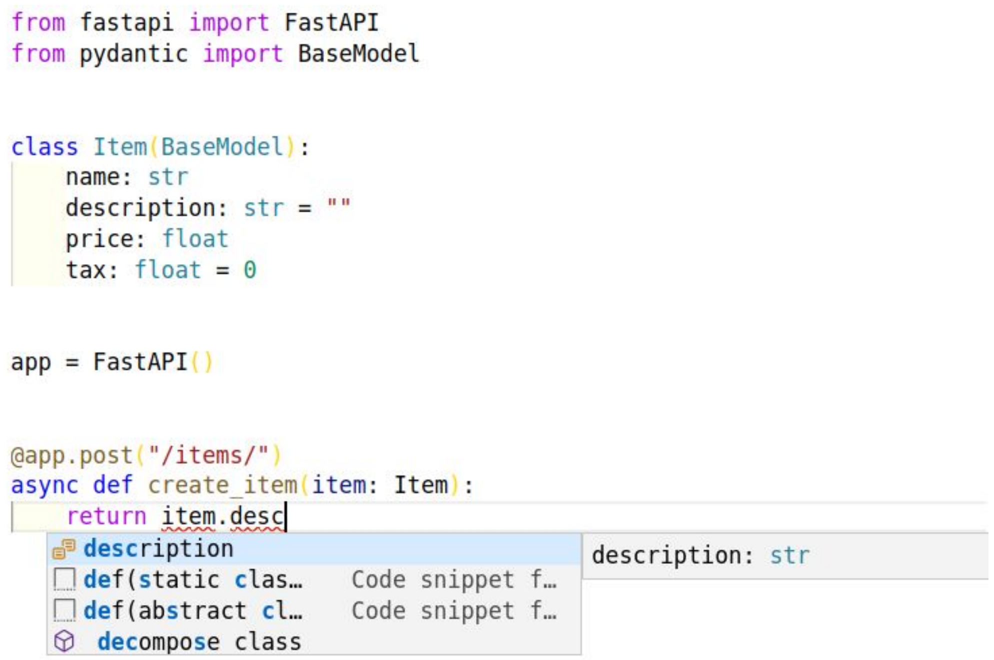
```
{
  "name": "string",
  "price": 0,
  "is_offer": true
}
```

@tiangolo

Based on standards - for free

Based on Python type hints

- Autocompletion everywhere
- Type checks and type errors

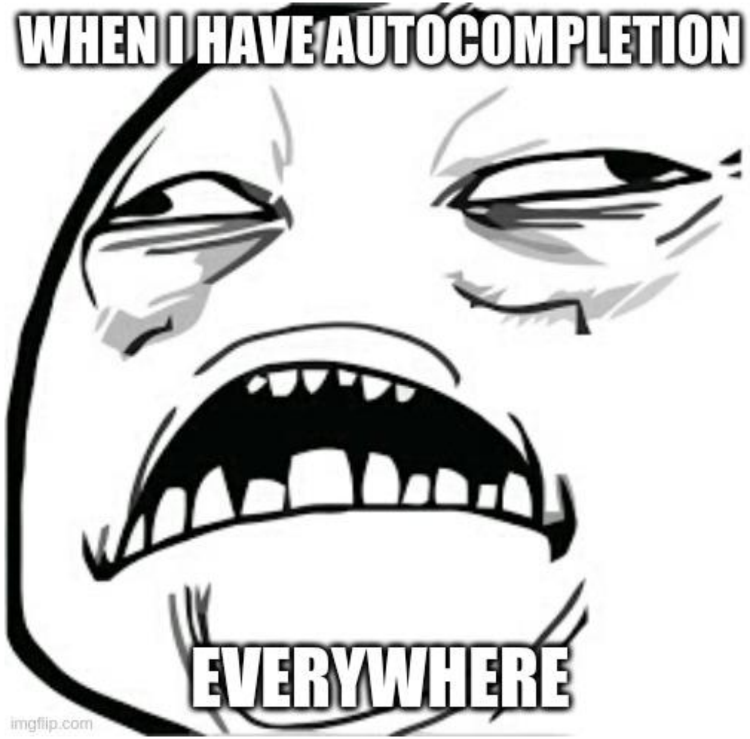


```python
from fastapi import FastAPI
from pydantic import BaseModel


class Item(BaseModel):
    name: str
    description: str = ""
    price: float
    tax: float = 0


app = FastAPI()


@app.post("/items/")
async def create_item(item: Item):
    return item.desc
```

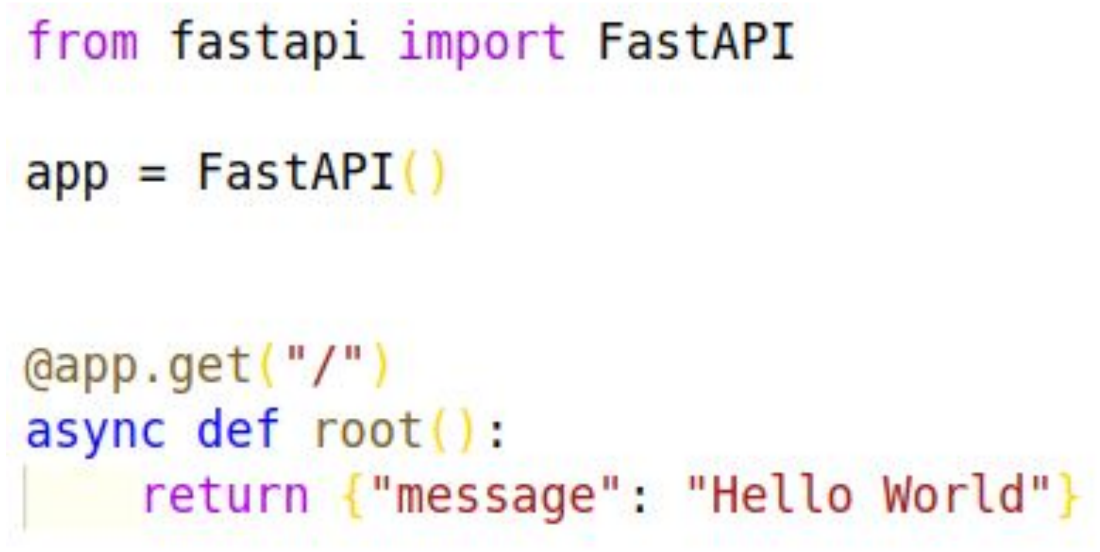
description
def(static clas… Code snippet f…
def(abstract cl… Code snippet f…
decompose class
description: str

@tiangolo

Types provide autocompletion

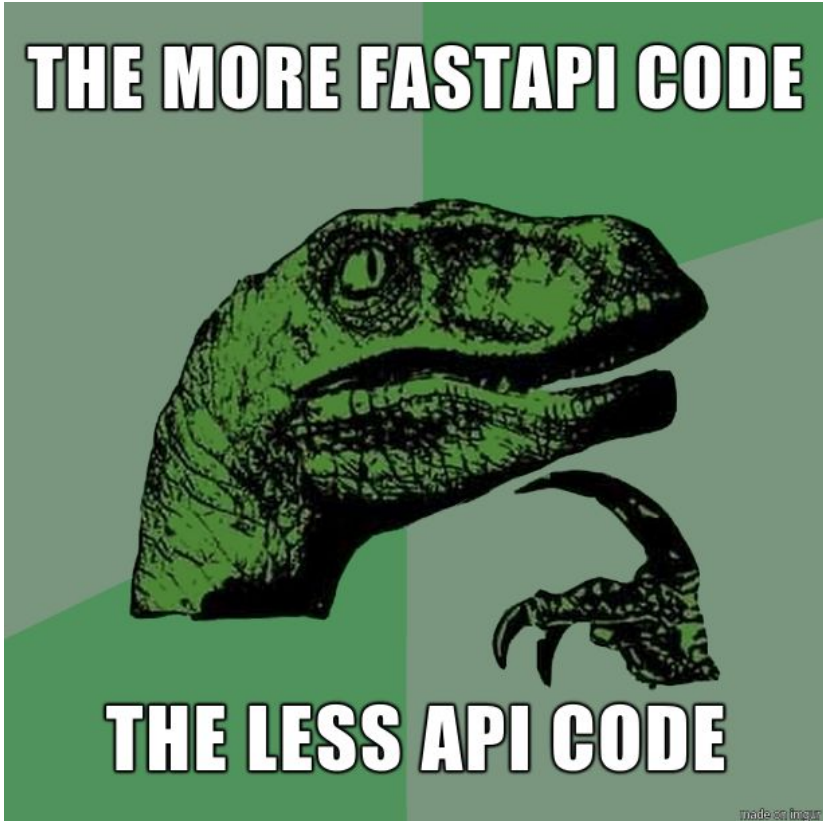


Easy and short



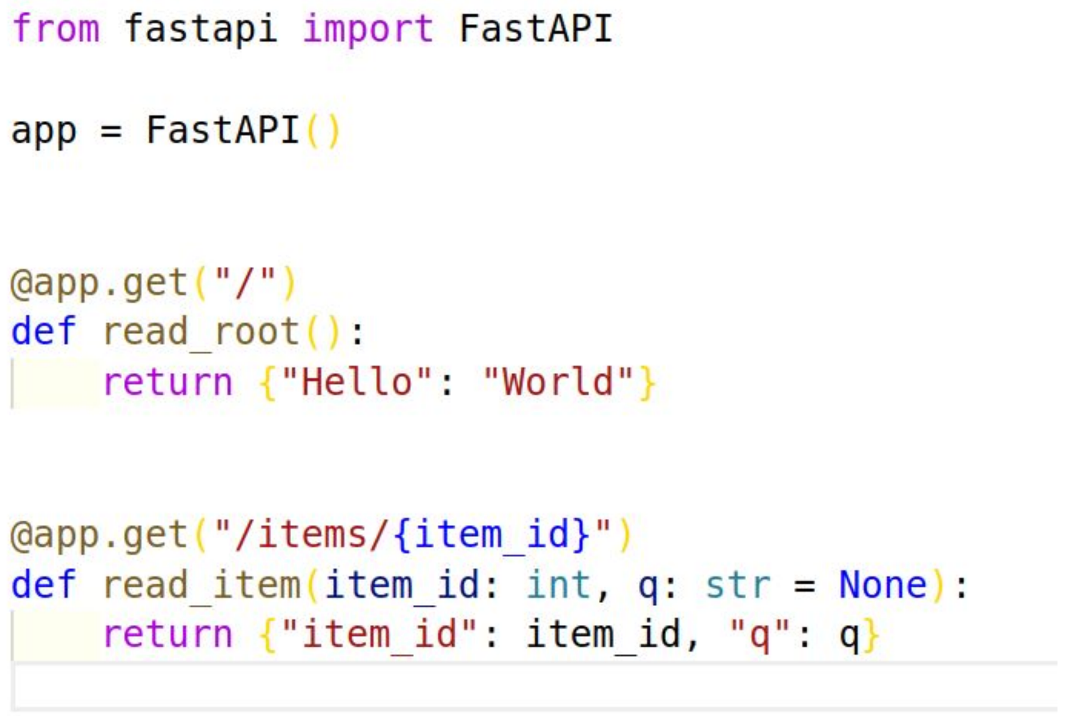
```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Less API code



@tiangolo

Basic **FastAPI** app

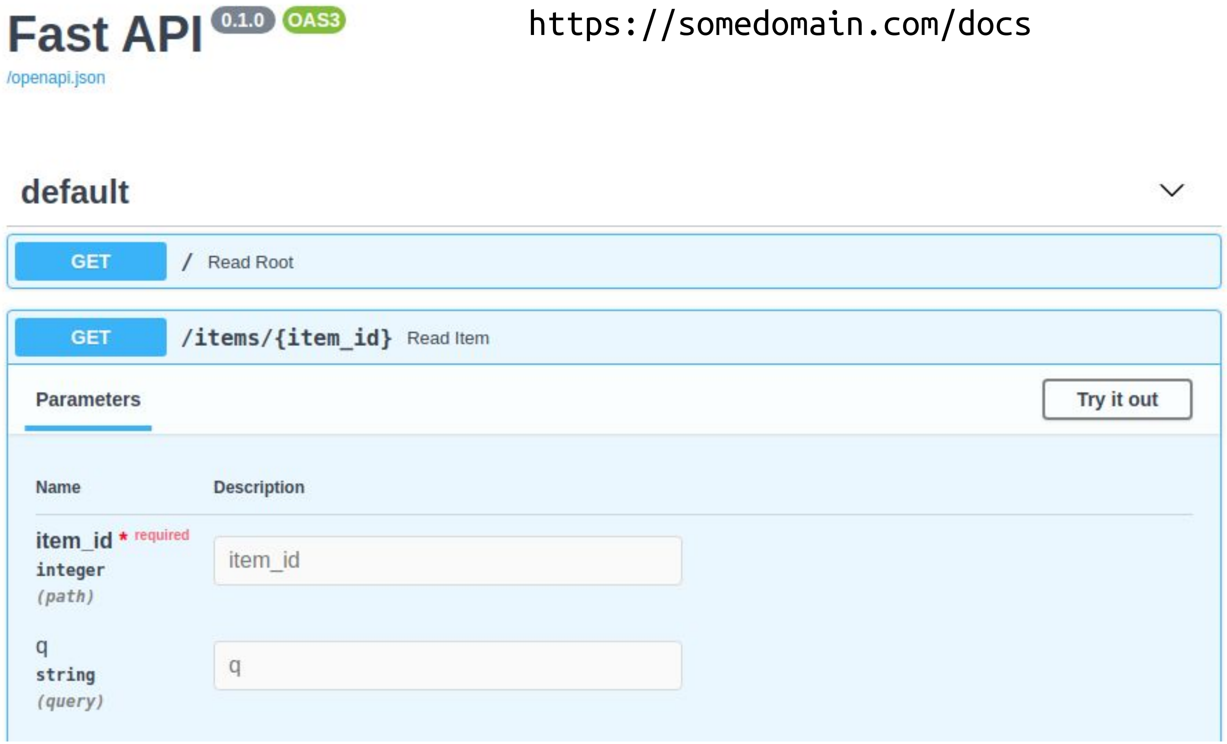


```python
from fastapi import FastAPI

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}
```

`https://somedomain.com/items/5?q=some+query`

@tiangolo

Basic **FastAPI** app - docs



Fast API 0.1.0 OAS3

/openapi.json

default

GET / Read Root

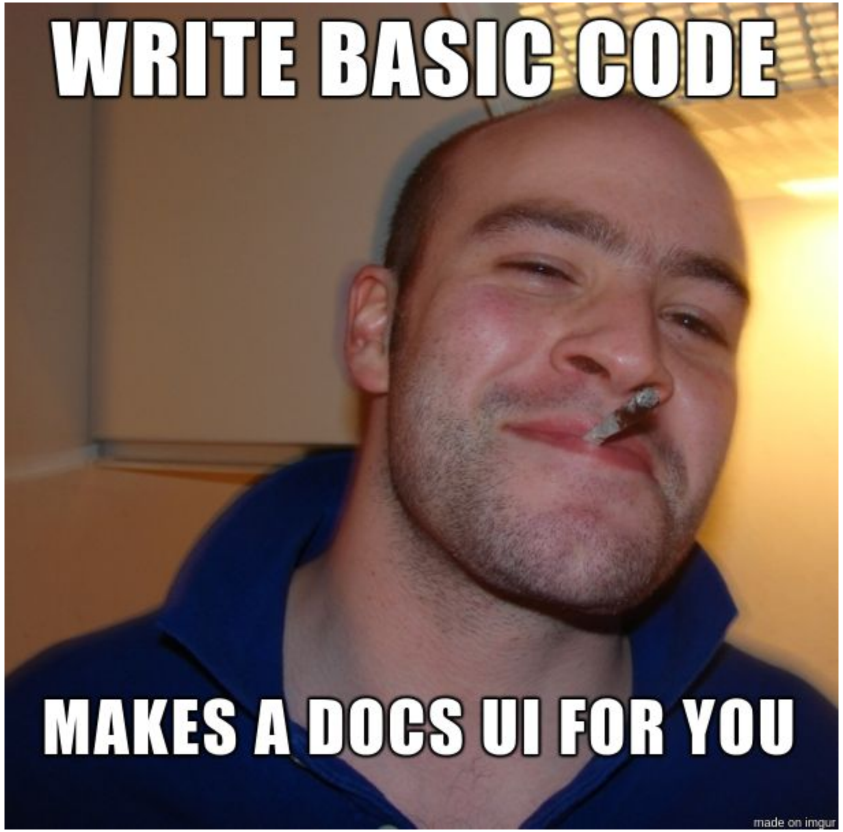
GET /items/{item_id} Read Item

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

https://somedomain.com/docs

@tiangolo

Automatic docs

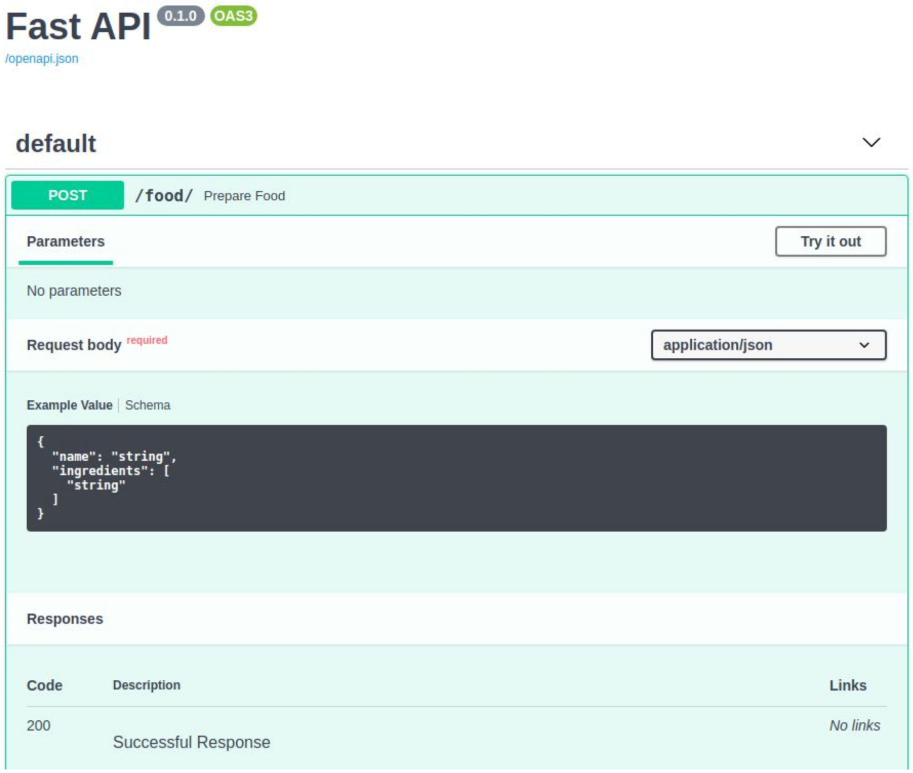


Basic **FastAPI** app with body

```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(food: Food):
    return {"message": f"preparing {food.name}"}
```

@tiangolo

Basic **FastAPI** app with body - docs

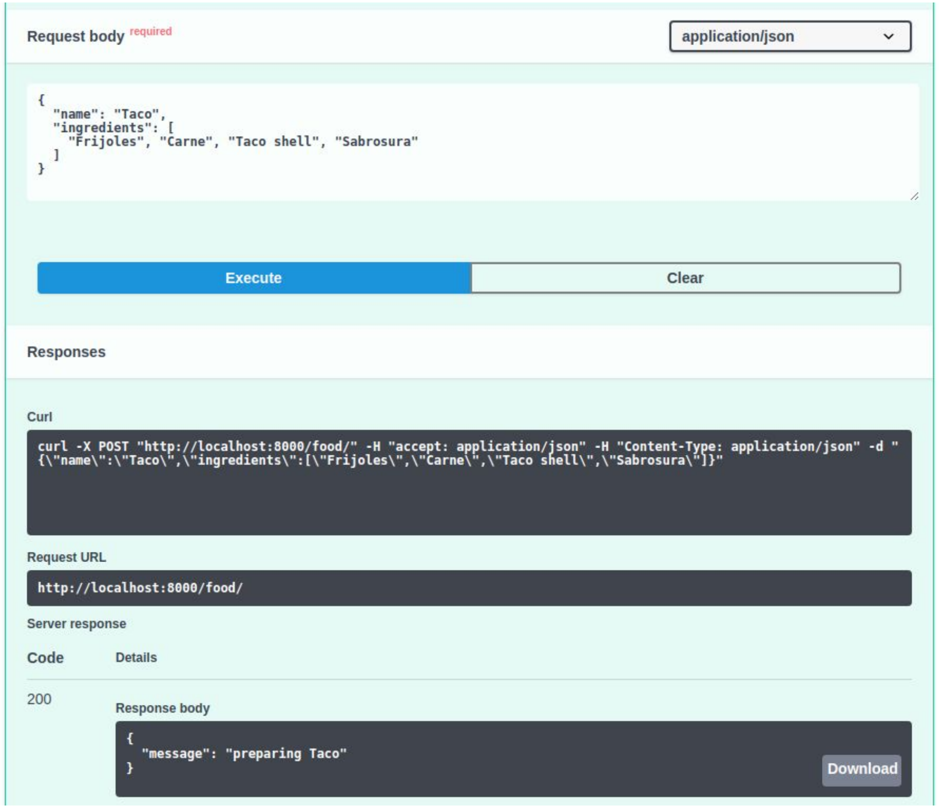


```
{
  "name": "string",
  "ingredients": [
    "string"
  ]
}
```

Code	Description	Links
200	Successful Response	No links

@tiangolo

Basic **FastAPI** app with body - docs



```
{
  "name": "Taco",
  "ingredients": [
    "Frijoles", "Carne", "Taco shell", "Sabrosura"
  ]
}
```

Curl

```
curl -X POST "http://localhost:8000/food/" -H "accept: application/json" -H "Content-Type: application/json" -d "
{\"name\":\"Taco\",\"ingredients\":[\"Frijoles\",\"Carne\",\"Taco shell\",\"Sabrosura\"]}"
```

Request URL

```
http://localhost:8000/food/
```

Server response

Code	Details
200	Response body

```
{
  "message": "preparing Taco"
}
```

@tiangolo

Basic **FastAPI** app with body

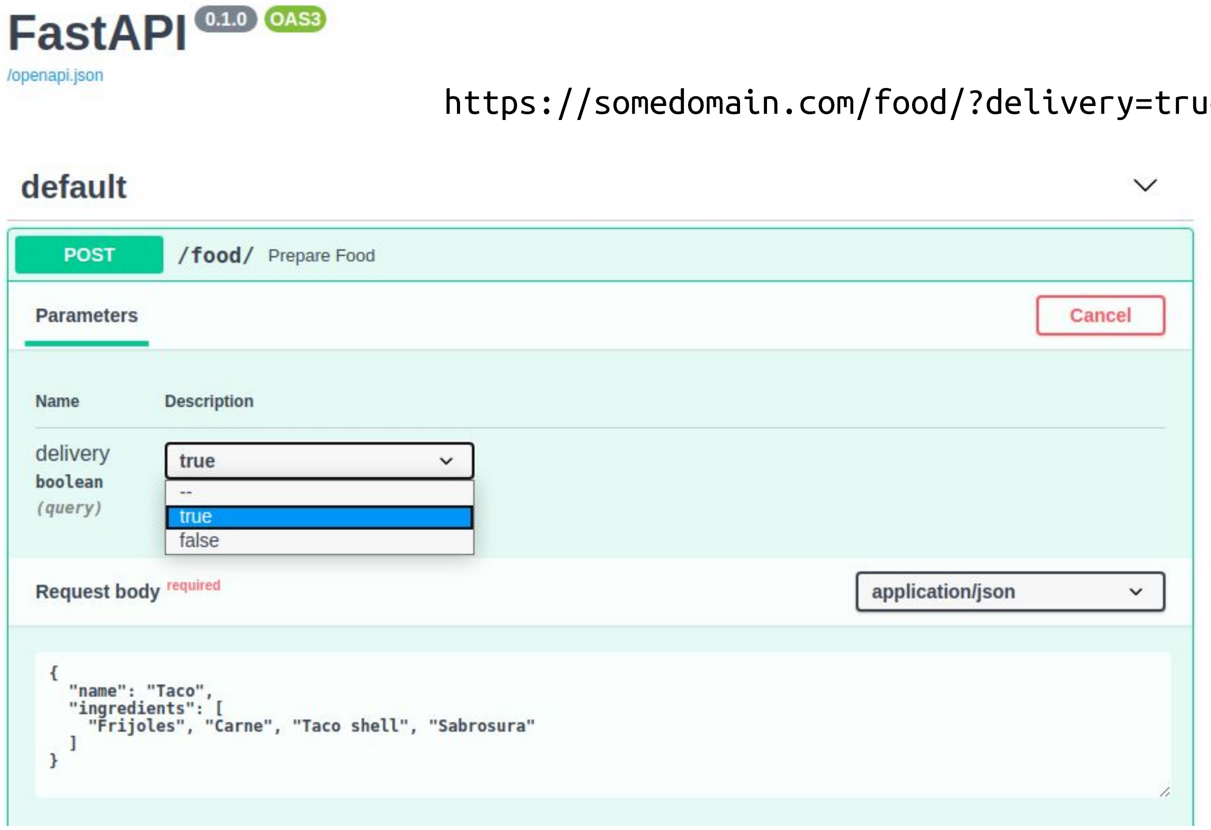
@tiangolo

Basic **FastAPI** app with body and query

```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(food: Food, delivery: bool = False):
    return {"message": f"preparing {food.name}", "delivery": delivery}
```

@tiangolo

Basic **FastAPI** app with body and query - docs



FastAPI 0.1.0 OAS3

/openapi.json

default

POST /food/ Prepare Food

Parameters — Cancel

Name	Description
delivery boolean *(query)*	true (-- / true / false)

Request body required — application/json

```
{
  "name": "Taco",
  "ingredients": [
    "Frijoles", "Carne", "Taco shell", "Sabrosura"
  ]
}
```

```
https://somedomain.com/food/?delivery=true
```

@tiangolo

Basic **FastAPI** app with body and query

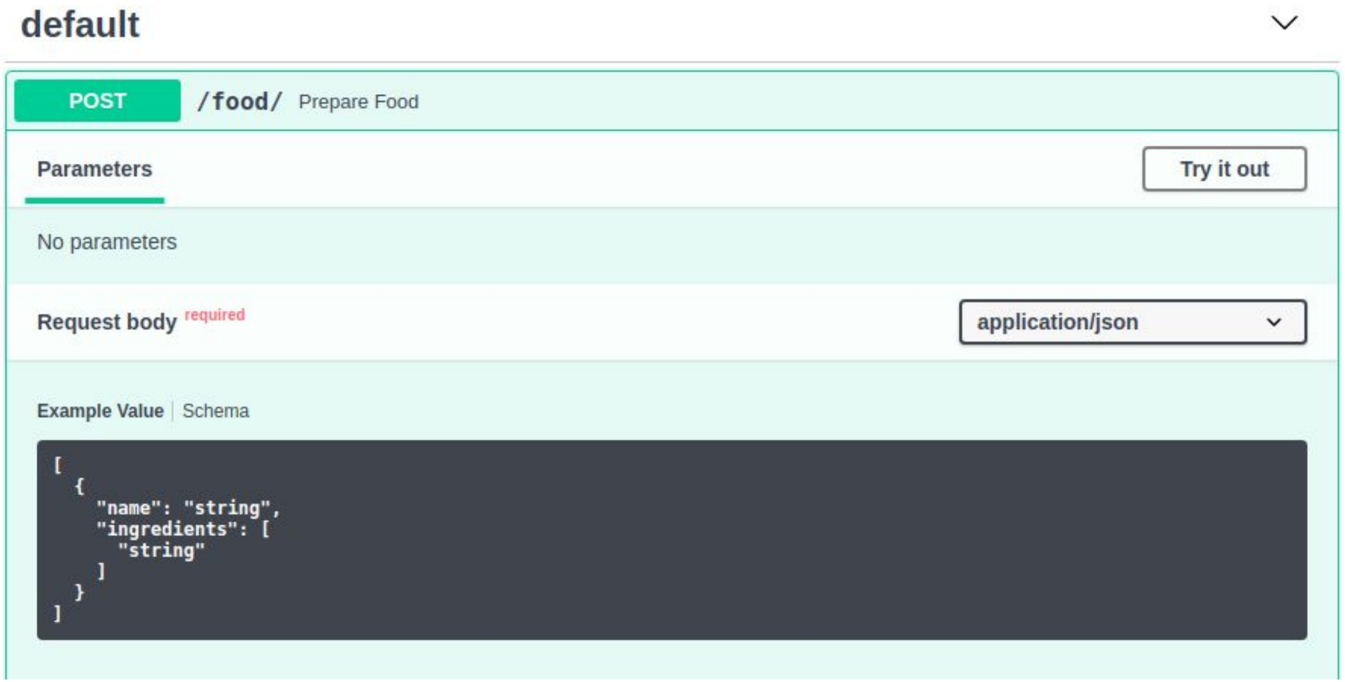
@tiangolo

JSON array of objects in body

```
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(orders: List[Food]):
    all_ingredients = []
    for food in orders:
        for ingredient in food.ingredients:
            all_ingredients.append(ingredient.lower())

    return {"ingredients": all_ingredients}
```

@tiangolo

JSON array of objects in body - docs


default

POST /food/ Prepare Food

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

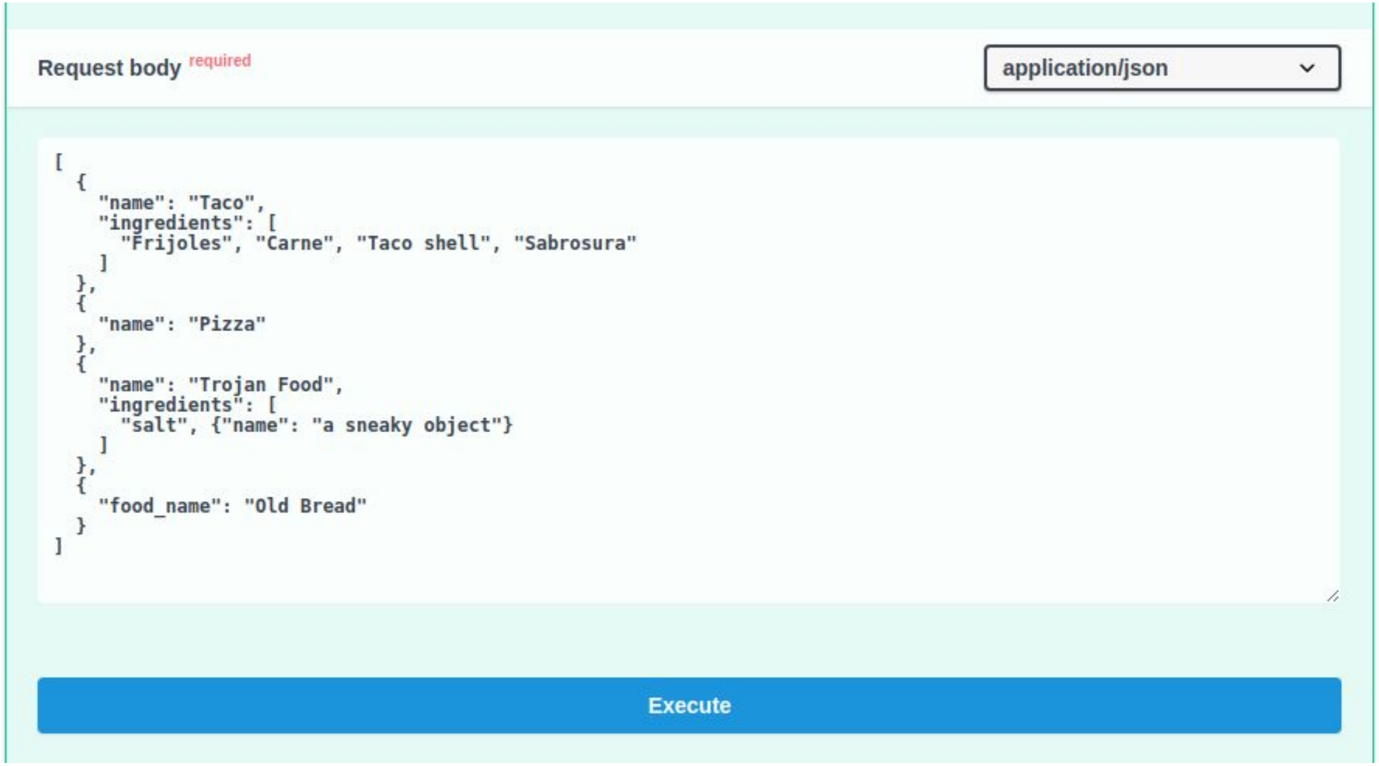
```
[
  {
    "name": "string",
    "ingredients": [
      "string"
    ]
  }
]
```

@tiangolo

JSON array of objects in body

@tiangolo

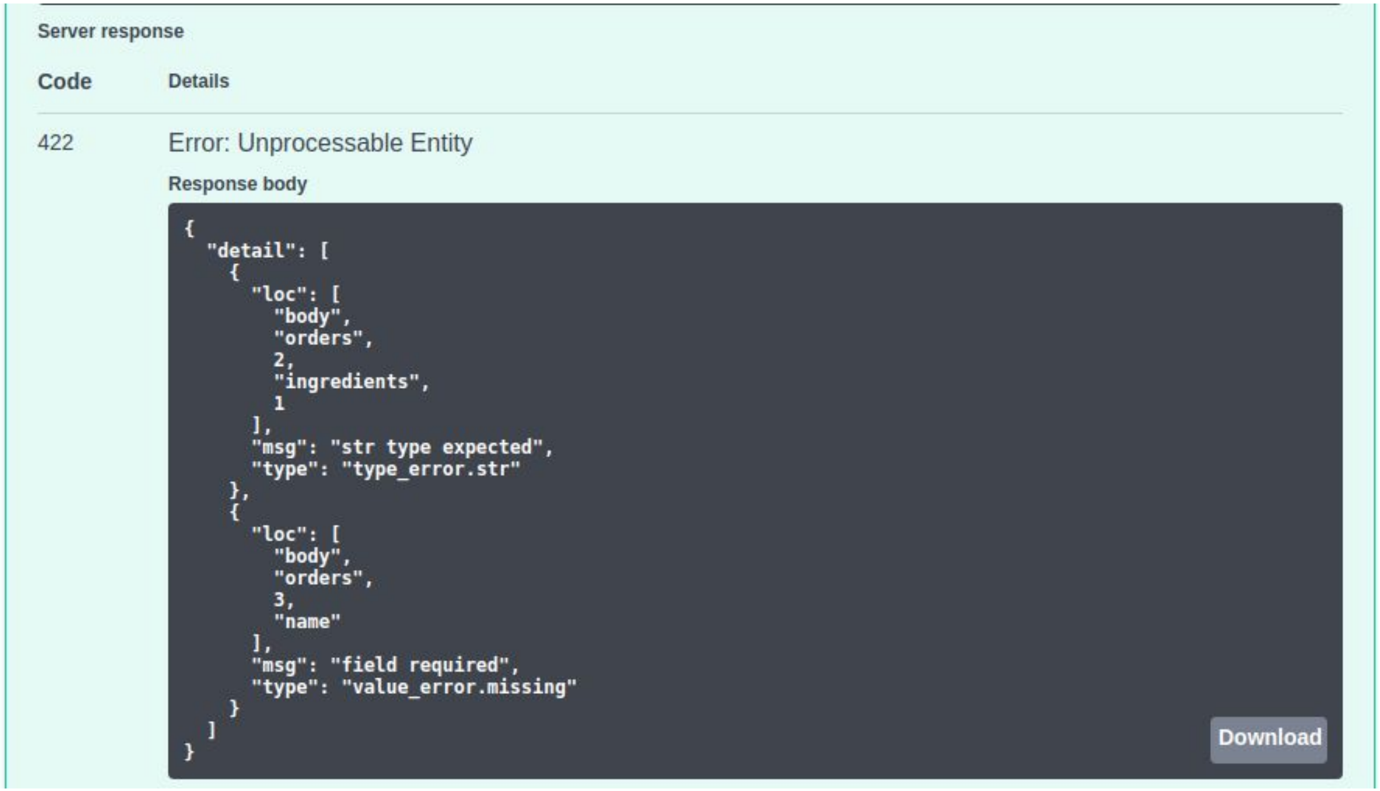
JSON array of objects in body - invalid data



```
[
  {
    "name": "Taco",
    "ingredients": [
      "Frijoles", "Carne", "Taco shell", "Sabrosura"
    ]
  },
  {
    "name": "Pizza"
  },
  {
    "name": "Trojan Food",
    "ingredients": [
      "salt", {"name": "a sneaky object"}
    ]
  },
  {
    "food_name": "Old Bread"
  }
]
```

@tiangolo

JSON array of objects in body - validation



Server response

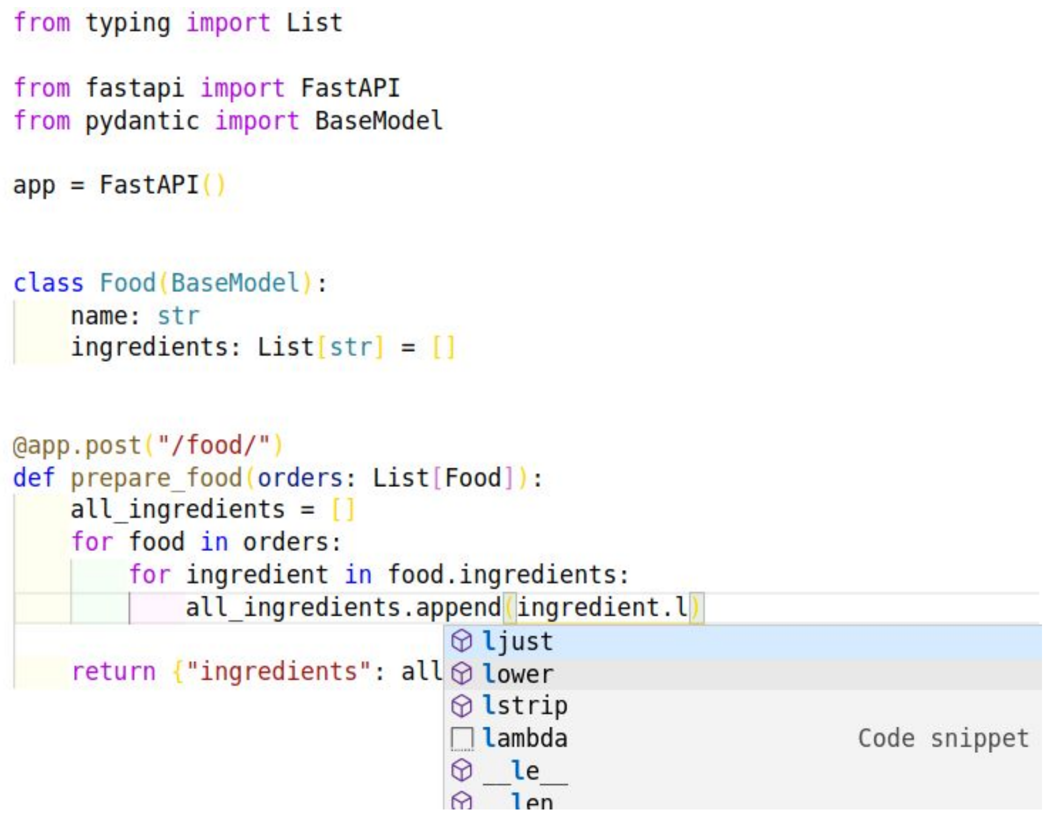
Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "loc": [
        "body",
        "orders",
        2,
        "ingredients",
        1
      ],
      "msg": "str type expected",
      "type": "type_error.str"
    },
    {
      "loc": [
        "body",
        "orders",
        3,
        "name"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    }
  ]
}
```

JSON array of objects in body - validation

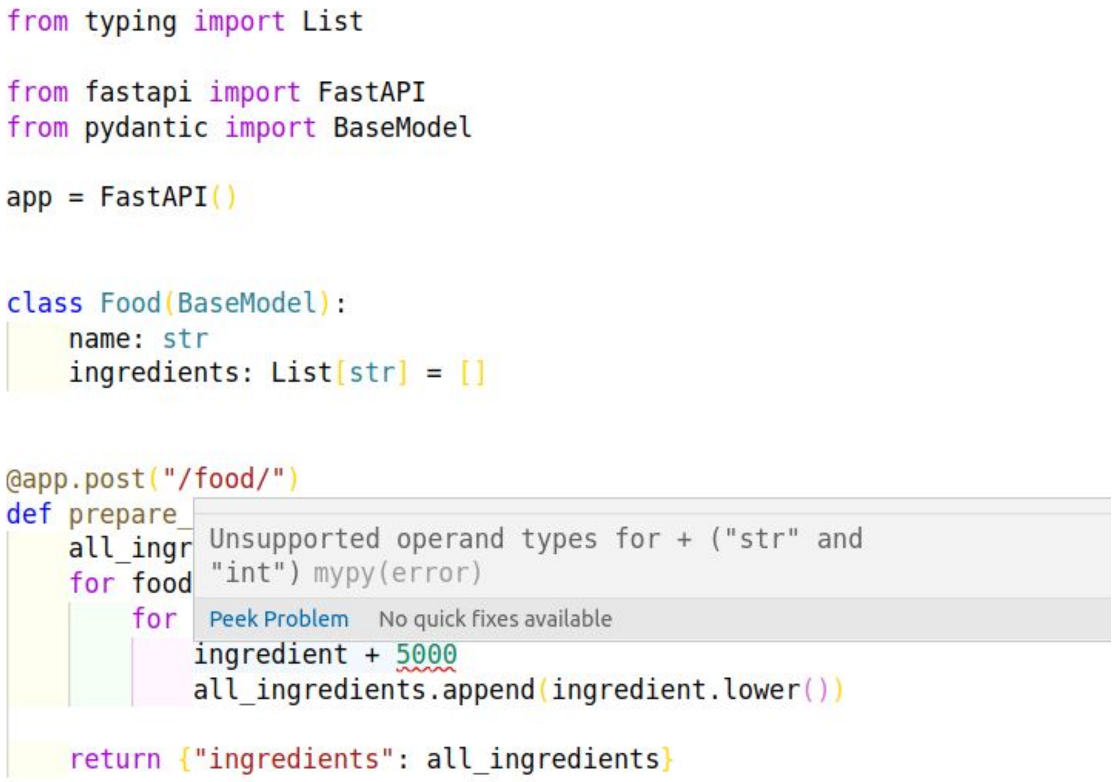
FastAPI auto-completion



```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(orders: List[Food]):
    all_ingredients = []
    for food in orders:
        for ingredient in food.ingredients:
            all_ingredients.append(ingredient.l)

    return {"ingredients": all
```

@tiangolo

Type checks



```
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_
    all_ingr
    for food
        for
            ingredient + 5000
            all_ingredients.append(ingredient.lower())

    return {"ingredients": all_ingredients}
```

```
Unsupported operand types for + ("str" and
"int") mypy(error)
```

@tiangolo

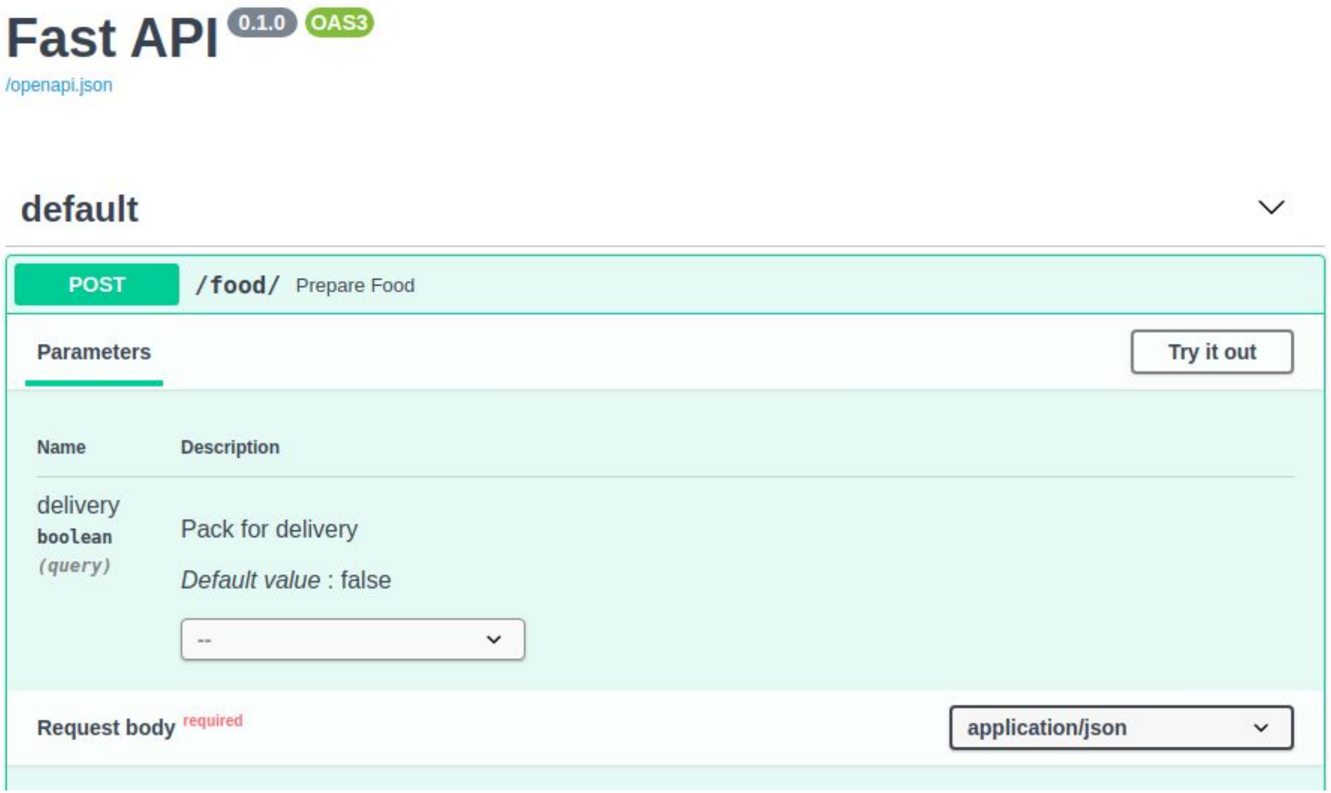
All based on standard Python types

Metadata

```
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(
    food: Food,
    delivery: bool = Query(False, description="Pack for delivery"),
):
    return {"message": f"preparing {food.name}", "delivery": delivery}
```

@tiangolo

Metadata in docs



Fast API 0.1.0 OAS3

/openapi.json

default

POST /food/ Prepare Food

Parameters

Try it out

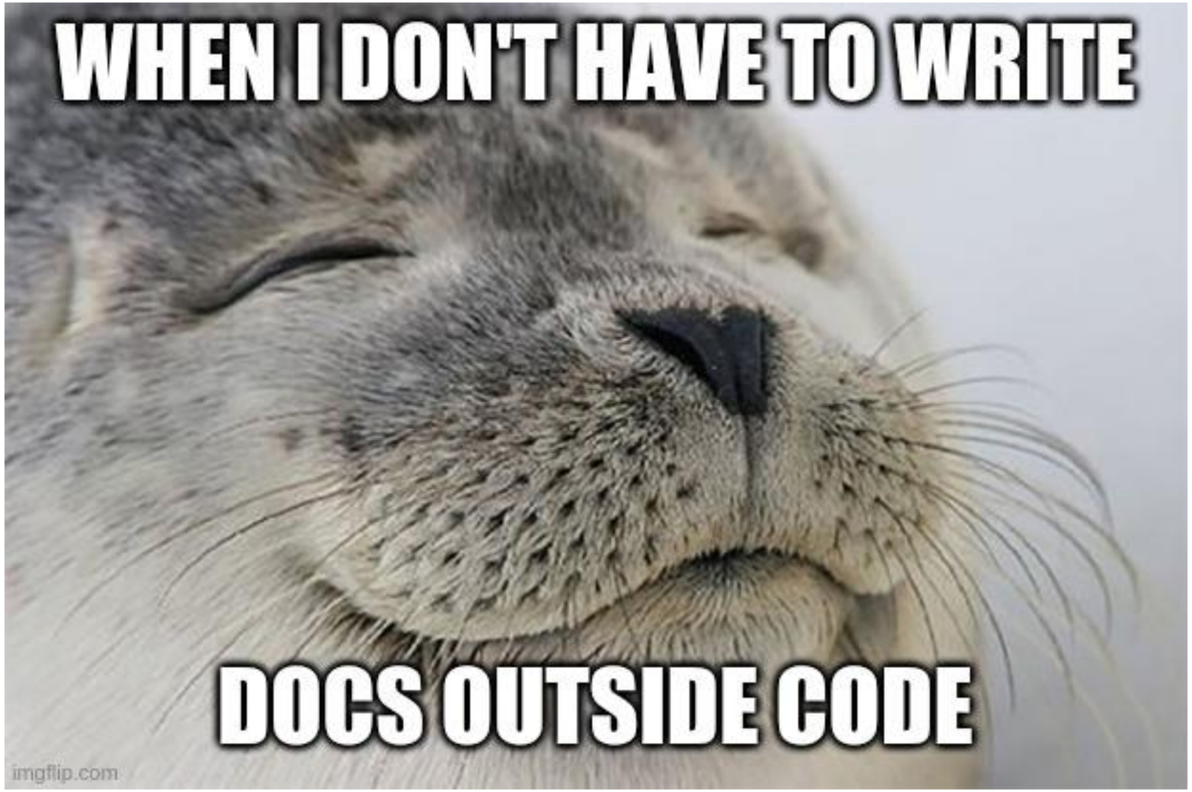
Name	Description
delivery boolean *(query)*	Pack for delivery *Default value* : false --

Request body required

application/json

@tiangolo

Metadata in docs

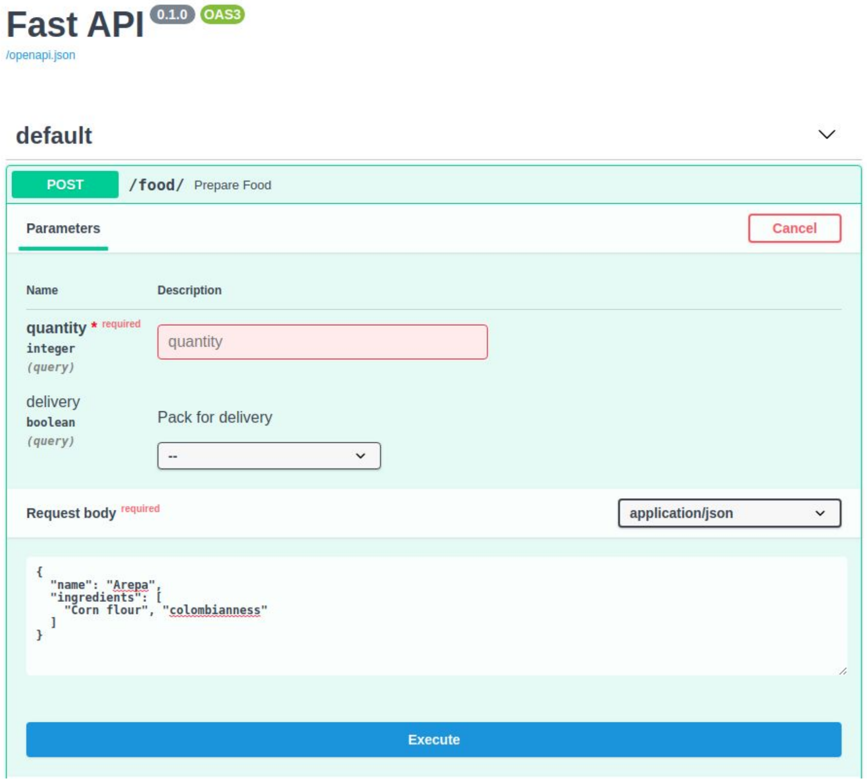


Required query

```python
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(
    food: Food,
    quantity: int,
    delivery: bool = Query(False, description="Pack for delivery"),
):
    return {
        "message": f"preparing {quantity} {food.name}",
        "delivery": delivery
    }
```

@tiangolo

Required query in docs



Fast API 0.1.0 OAS3
/openapi.json

default

POST /food/ Prepare Food

Parameters   Cancel

Name	Description
quantity * required integer *(query)*	quantity
delivery boolean *(query)*	Pack for delivery --

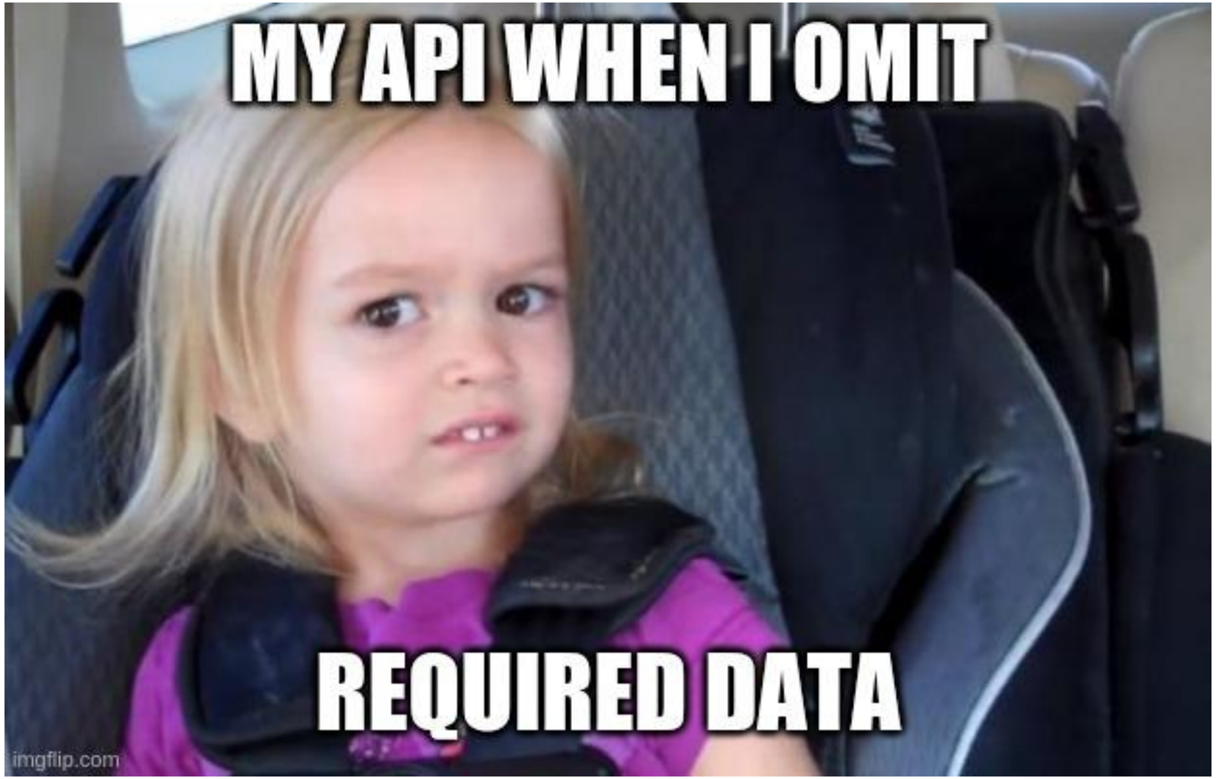
Request body required   application/json

```
{
  "name": "Arepa",
  "ingredients": [
    "Corn flour", "colombianness"
  ]
}
```

Execute

@tiangolo

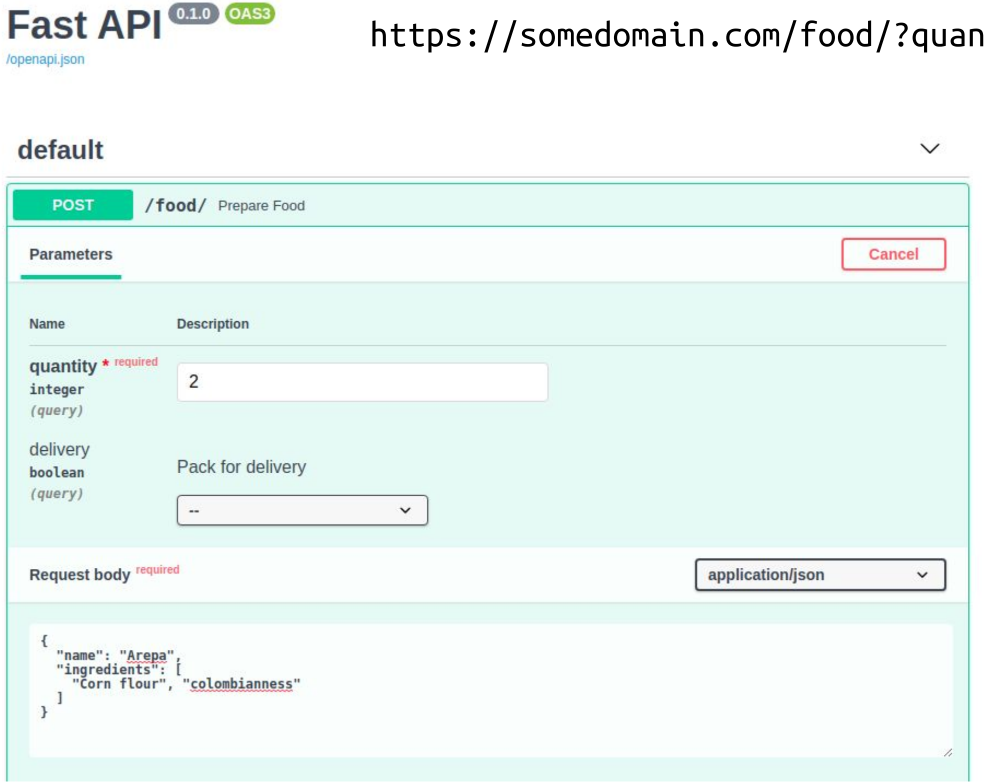
Required query validation



@tiangolo

Required query in docs - valid

`https://somedomain.com/food/?quantity=2`



Fast API 0.1.0 OAS3
/openapi.json

default

POST /food/ Prepare Food

Parameters

Cancel

Name	Description
quantity * required integer (query)	2
delivery boolean (query)	Pack for delivery --

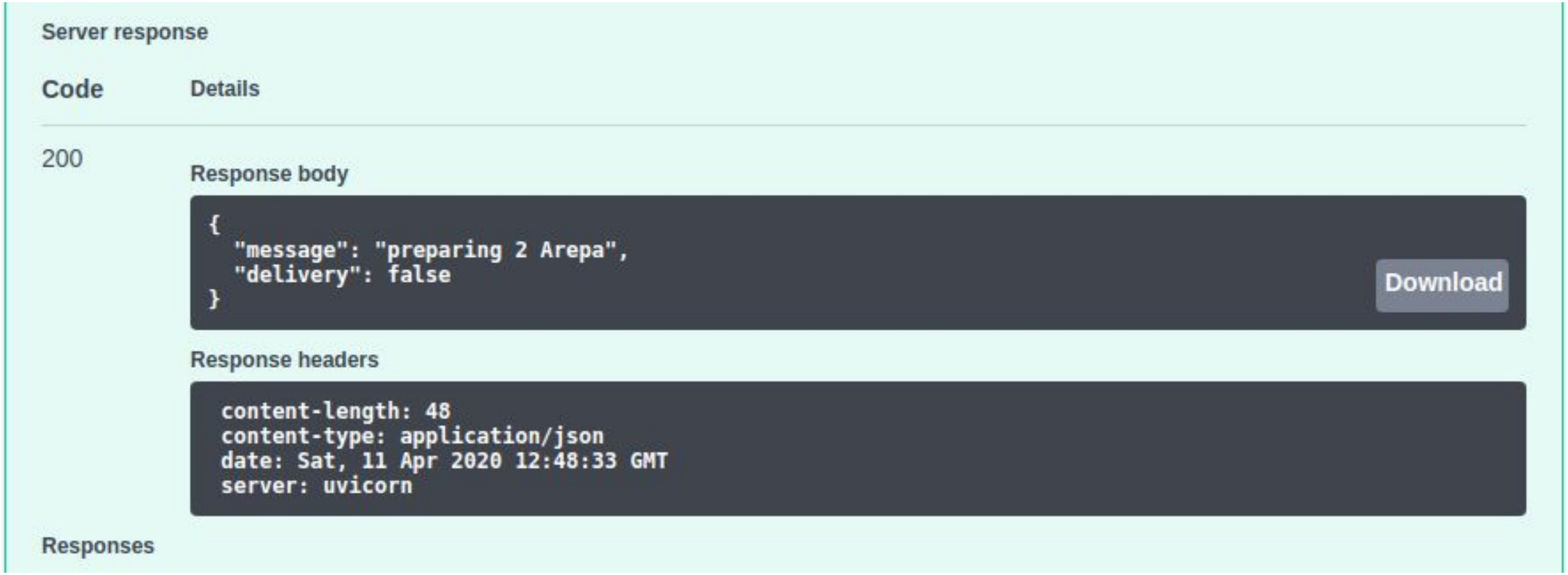
Request body required

application/json

```
{
  "name": "Arepa",
  "ingredients": [
    "Corn flour", "colombianness"
  ]
}
```

@tiangolo

Required query in docs - response



Server response

Code | Details

200

Response body

```
{
  "message": "preparing 2 Arepa",
  "delivery": false
}
```

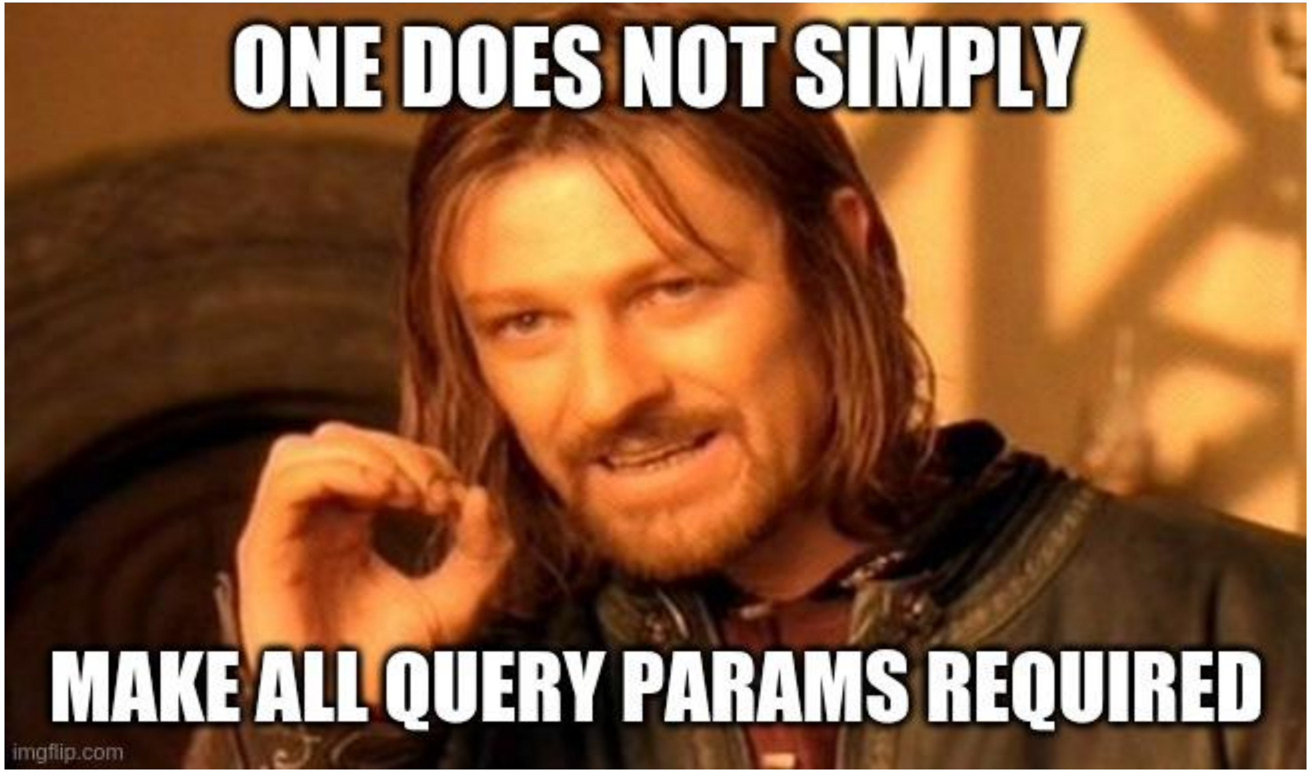
Response headers

```
 content-length: 48
 content-type: application/json
 date: Sat, 11 Apr 2020 12:48:33 GMT
 server: uvicorn
```

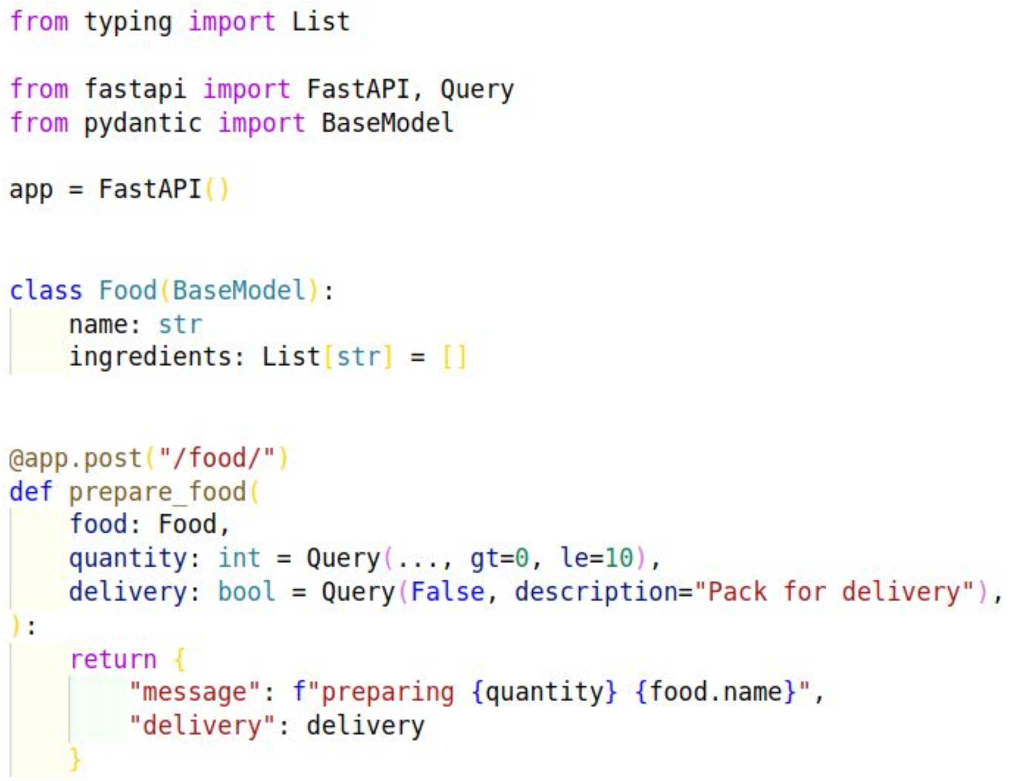
Responses

@tiangolo

Required query in docs

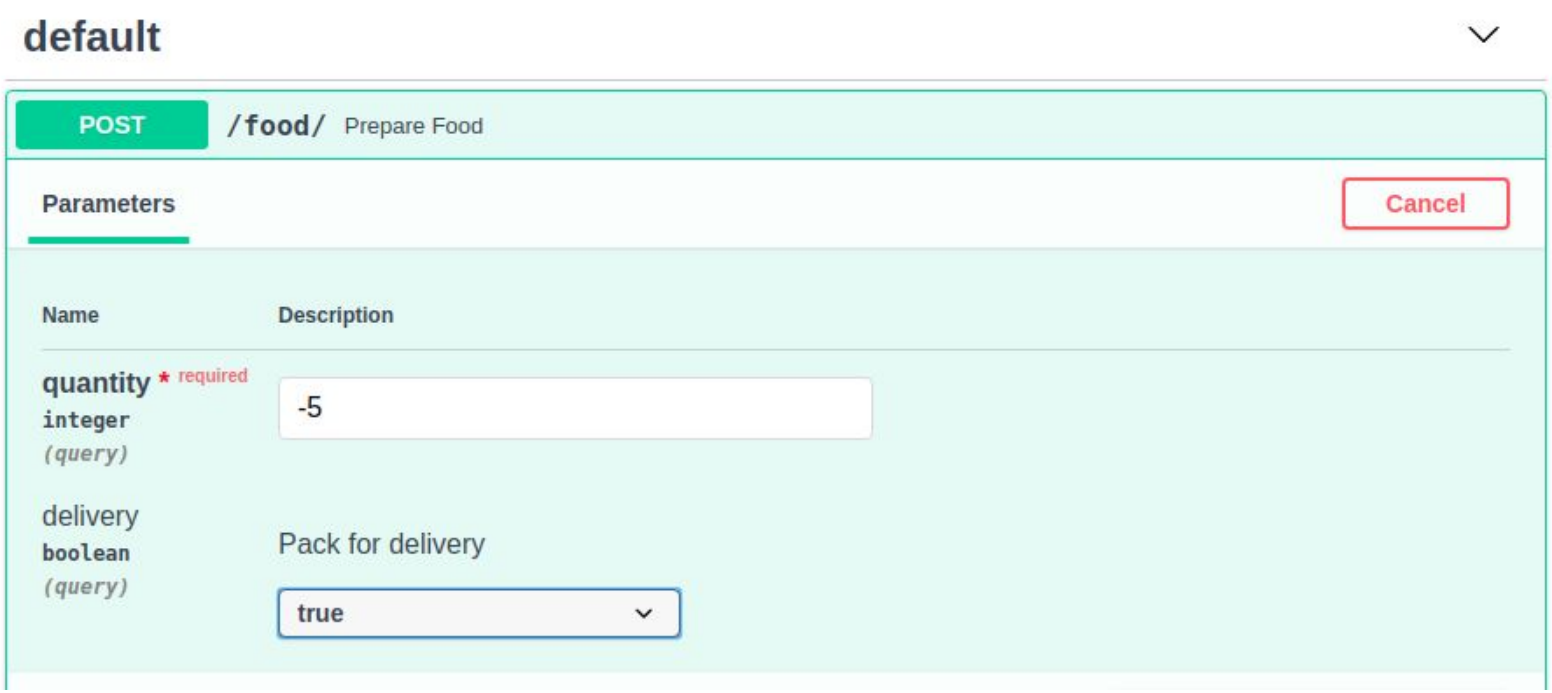


Validation



```python
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel

app = FastAPI()


class Food(BaseModel):
    name: str
    ingredients: List[str] = []


@app.post("/food/")
def prepare_food(
    food: Food,
    quantity: int = Query(..., gt=0, le=10),
    delivery: bool = Query(False, description="Pack for delivery"),
):
    return {
        "message": f"preparing {quantity} {food.name}",
        "delivery": delivery
    }
```

Validation in docs



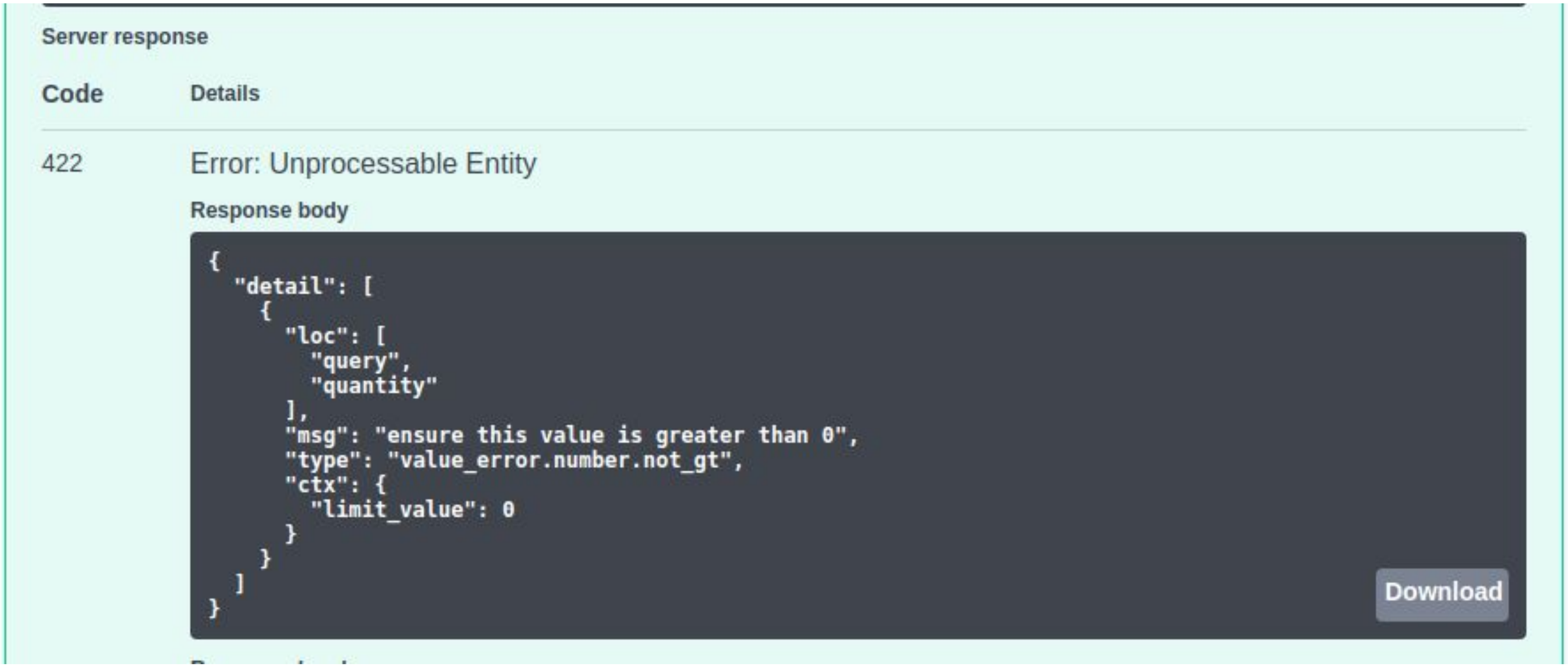
/food/ Prepare Food

Parameters

Name	Description
quantity * required integer (query)	-5
delivery boolean (query)	Pack for delivery: true

@tiangolo

Validation in docs - response



Server response

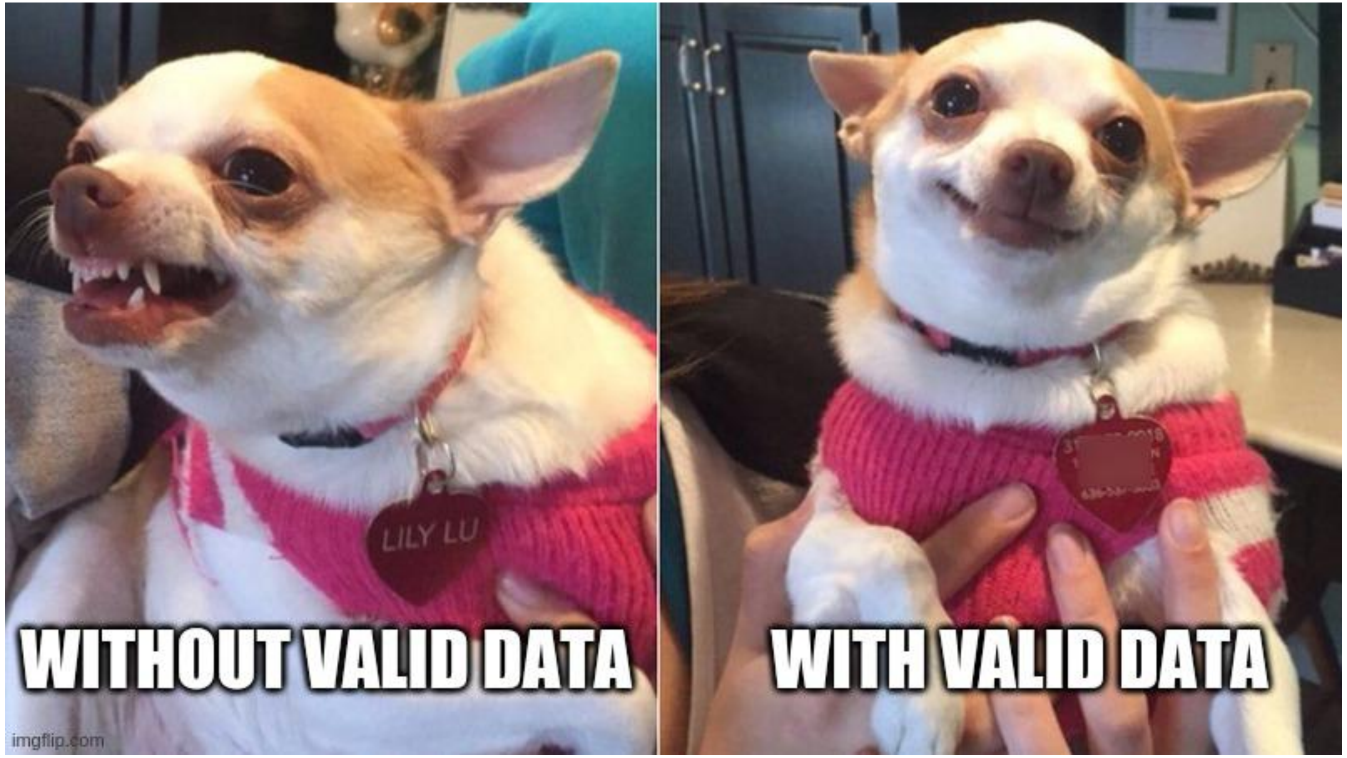
Code	Details
422	Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "loc": [
        "query",
        "quantity"
      ],
      "msg": "ensure this value is greater than 0",
      "type": "value_error.number.not_gt",
      "ctx": {
        "limit_value": 0
      }
    }
  ]
}
```

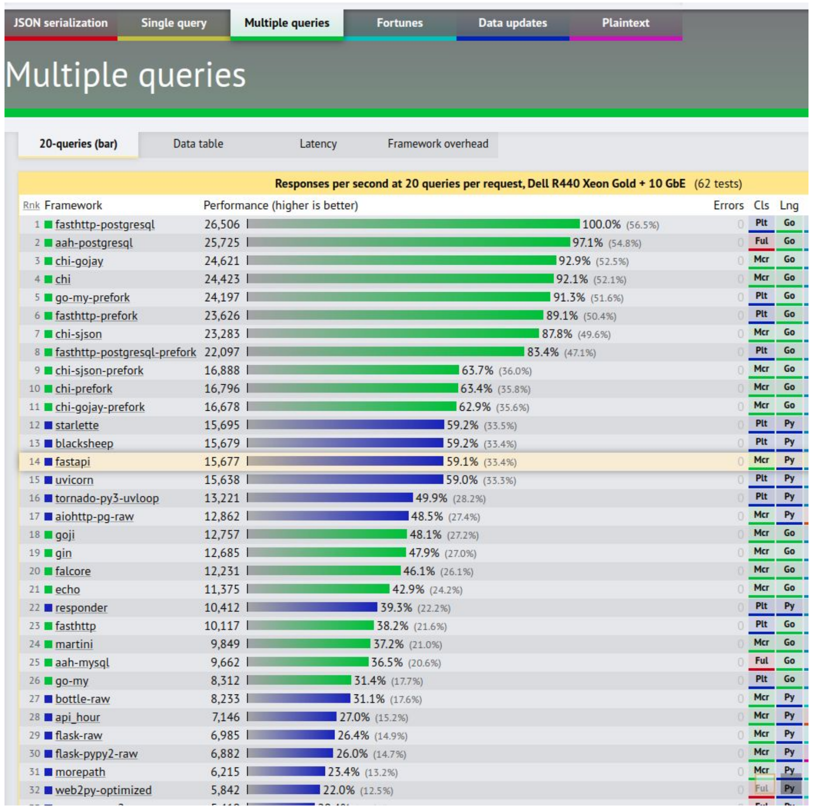
@tiangolo

Data validation



@tiangolo

FastAPI - Performance



JSON serialization | Single query | Multiple queries | Fortunes | Data updates | Plaintext

Multiple queries

20-queries (bar) | Data table | Latency | Framework overhead

Responses per second at 20 queries per request, Dell R440 Xeon Gold + 10 GbE (62 tests)

Rnk	Framework	Performance (higher is better)			Errors	Cls	Lng
1	🟩 fasthttp-postgresql	26,506	100.0%	(56.5%)	0	Plt	Go
2	🟩 aah-postgresql	25,725	97.1%	(54.8%)	0	Ful	Go
3	🟩 chi-gojay	24,621	92.9%	(52.5%)	0	Mcr	Go
4	🟩 chi	24,423	92.1%	(52.1%)	0	Mcr	Go
5	🟩 go-my-prefork	24,197	91.3%	(51.6%)	0	Plt	Go
6	🟩 fasthttp-prefork	23,626	89.1%	(50.4%)	0	Plt	Go
7	🟩 chi-sjson	23,283	87.8%	(49.6%)	0	Mcr	Go
8	🟩 fasthttp-postgresql-prefork	22,097	83.4%	(47.1%)	0	Plt	Go
9	🟩 chi-sjson-prefork	16,888	63.7%	(36.0%)	0	Mcr	Go
10	🟩 chi-prefork	16,796	63.4%	(35.8%)	0	Mcr	Go
11	🟩 chi-gojay-prefork	16,678	62.9%	(35.6%)	0	Mcr	Go
12	🟦 starlette	15,695	59.2%	(33.5%)	0	Plt	Py
13	🟦 blacksheep	15,679	59.2%	(33.4%)	0	Plt	Py
14	🟦 fastapi	15,677	59.1%	(33.4%)	0	Mcr	Py
15	🟦 uvicorn	15,638	59.0%	(33.3%)	0	Plt	Py
16	🟦 tornado-py3-uvloop	13,221	49.9%	(28.2%)	0	Plt	Py
17	🟦 aiohttp-pg-raw	12,862	48.5%	(27.4%)	0	Mcr	Py
18	🟩 goji	12,757	48.1%	(27.2%)	0	Mcr	Go
19	🟩 gin	12,685	47.9%	(27.0%)	0	Mcr	Go
20	🟩 falcore	12,231	46.1%	(26.1%)	0	Mcr	Go
21	🟩 echo	11,375	42.9%	(24.2%)	0	Mcr	Go
22	🟦 responder	10,412	39.3%	(22.2%)	0	Plt	Py
23	🟩 fasthttp	10,117	38.2%	(21.6%)	0	Plt	Go
24	🟩 martini	9,849	37.2%	(21.0%)	0	Mcr	Go
25	🟩 aah-mysql	9,662	36.5%	(20.6%)	0	Ful	Go
26	🟩 go-my	8,312	31.4%	(17.7%)	0	Plt	Go
27	🟦 bottle-raw	8,233	31.1%	(17.6%)	0	Mcr	Py
28	🟦 api_hour	7,146	27.0%	(15.2%)	0	Mcr	Py
29	🟦 flask-raw	6,985	26.4%	(14.9%)	0	Mcr	Py
30	🟦 flask-pypy2-raw	6,882	26.0%	(14.7%)	0	Mcr	Py
31	🟦 morepath	6,215	23.4%	(13.2%)	0	Mcr	Py
32	🟦 web2py-optimized	5,842	22.0%	(12.5%)	0	Ful	Py

@tiangolo

FastAPI - Performance

@tiangolo

FastAPI - Other features

- Dependency Injection
- Security - OAuth2
- WebSockets
- Files
- Background Tasks
- Easy GraphQL integration
- More...

FastAPI - When not to

- Stable
- No new features
- Not an API

Other tools

Typer

Build great CLIs. Easy to code.
Based on Python type hints.

typer.tiangolo.com

THiNC

Functional deep learning with types,
compatible with your favorite libraries.

thinc.ai

Thank you!

fastapi.tiangolo.com

Questions?

Sebastián Ramírez

tiangolo.com

github.com/tiangolo

linkedin.com/in/tiangolo

twitter.com/tiangolo