Getting Started with FastAPI for Web Development

Mia Bajić

About me

- software engineer with over 7 years of experience in the IT industry

About me

- software engineer with over 7 years of experience in the IT industry
- based in Prague, Czech Republic

About me

- software engineer with over 7 years of experience in the IT industry
- based in Prague, Czech Republic
- co-organizer of Prague Python meetups, Prague Python Pizza, PyCon CZ, EuroPython

My personal experience with Python frameworks

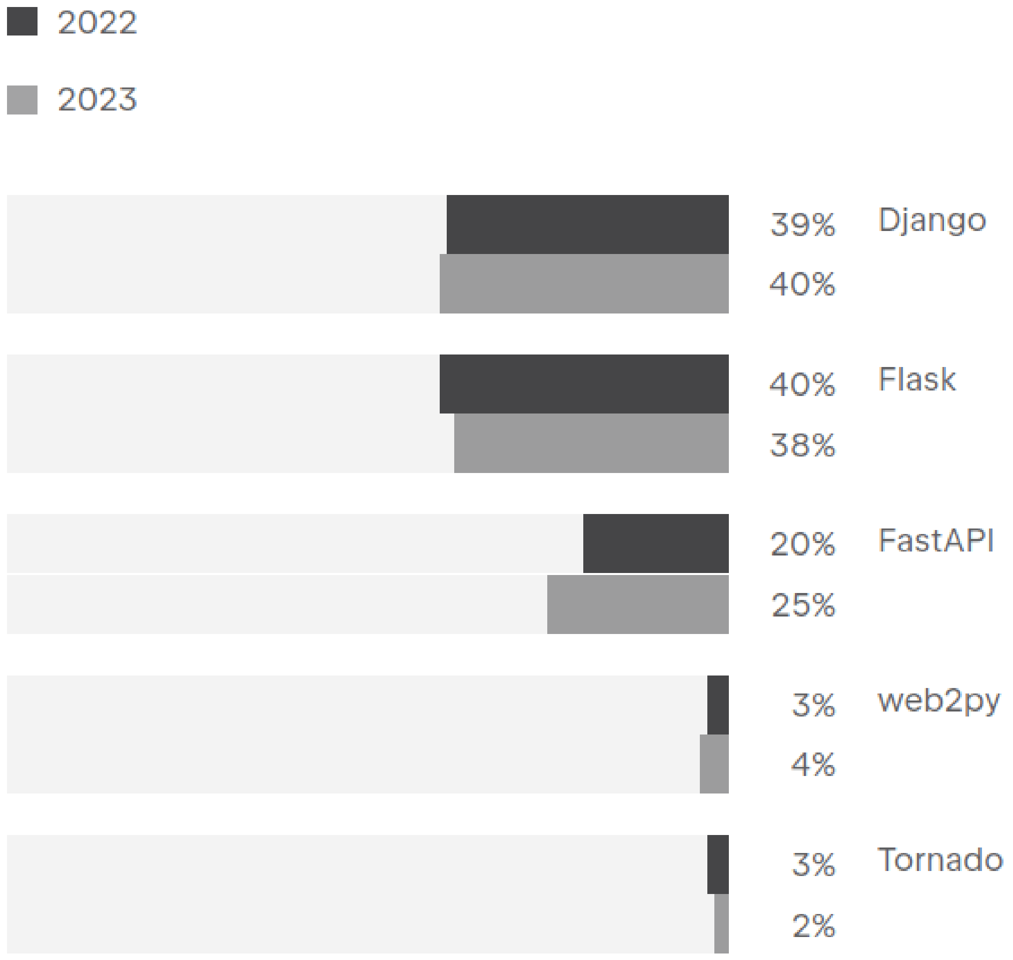
$$\begin{tikzcd}\text{Flask} \arrow[r] & \text{Django} \arrow[r] & \text{FastAPI}\end{tikzcd}$$

JetBrains Survey

JetBrains Survey

What web frameworks / libraries do you use in addition to Python?

JetBrains Survey



JetBrains Survey

- FastAPI has seen increasing usage over the past couple of years, rising from 14% in 2021 to 25% in 2023.

Aim of the talk

- Introduction to FastAPI

Aim of the talk

- Introduction to FastAPI
- Overview of key features

Aim of the talk

- Introduction to FastAPI
- Overview of key features
- Demo application

Aim of the talk

- Introduction to FastAPI
- Overview of key features
- Demo application
- Benefits of FastAPI

Aim of the talk

- Introduction to FastAPI
- Overview of key features
- Demo application
- Benefits of FastAPI
- Considerations for not using FastAPI

Introduction to FastAPI

- Framework for building REST APIs in Python

Introduction to FastAPI

- Framework for building REST APIs in Python
- Released in December 2018

Introduction to FastAPI

- Framework for building REST APIs in Python
- Released in December 2018
- Used in many world-known companies such as Netflix, Microsoft or Uber

Overview of key features

- Built on Starlette and Pydantic

Starlette

- Lightweight asynchronous framework or toolkit used for building web services

Starlette

- Lightweight asynchronous framework or toolkit used for building web services
- Any of its components can be used independently

Starlette

- Lightweight asynchronous framework or toolkit used for building web services
- Any of its components can be used independently
- Main features:
    - lightweight HTTP web framework
    - WebSocket, Session & Cookie support
    - Test client
    - CORS, GZip, Static files, Streaming responses
    - Background Tasks

Pydantic

- Python package for data validation

Pydantic

- Python package for data validation
- It checks data types of input and output data and returns errors if passed data is invalid

Pydantic

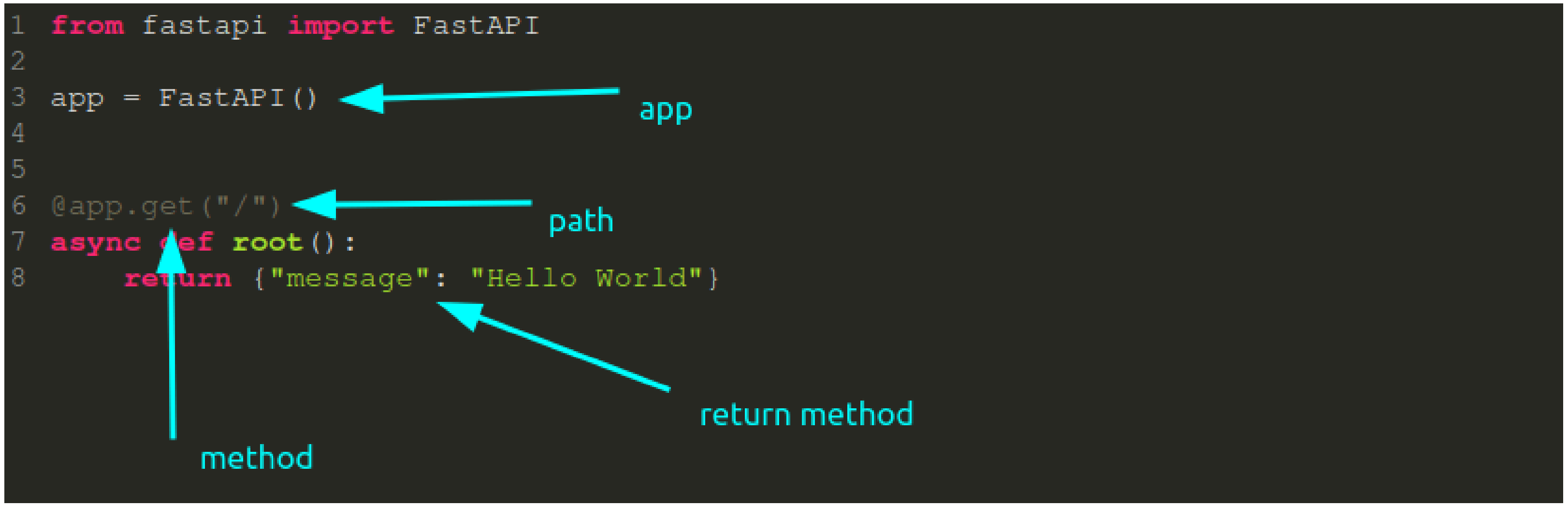
```python
from pydantic import BaseModel, validator

class User(BaseModel):
    username: str
    password: str
    age: int


    @validator('age')
    def age_must_be_over_eighteen(cls, v):
        if age < 18:
            raise ValueError('User must be at least 18 years old in order to be registered.')
            return v
```

FastAPI demo - let's make the simplest API

FastAPI demo - let's make the simplest API

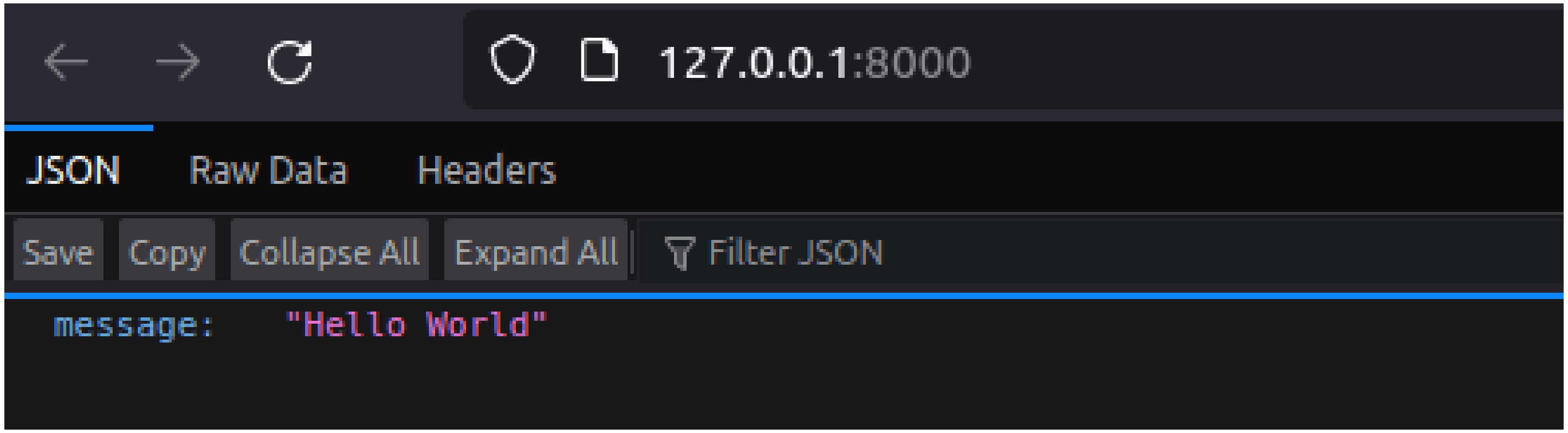
```python
from fastapi import FastAPI

app = FastAPI()


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

FastAPI demo - let's make the simplest API



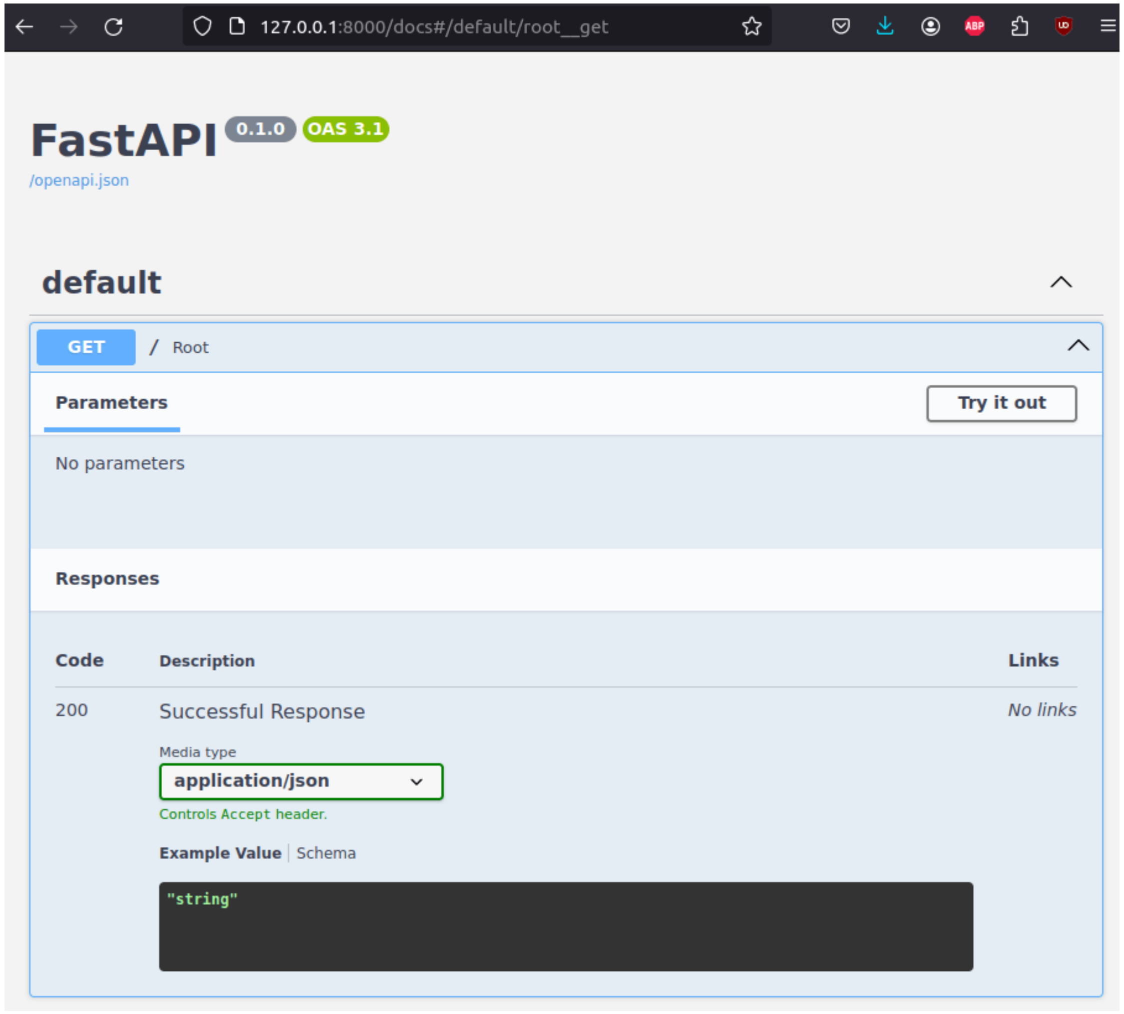
```
from fastapi import FastAPI

app = FastAPI()


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

FastAPI demo - let's make the simplest API



```
message:   "Hello World"
```

FastAPI demo - let's make the simplest API



No parameters

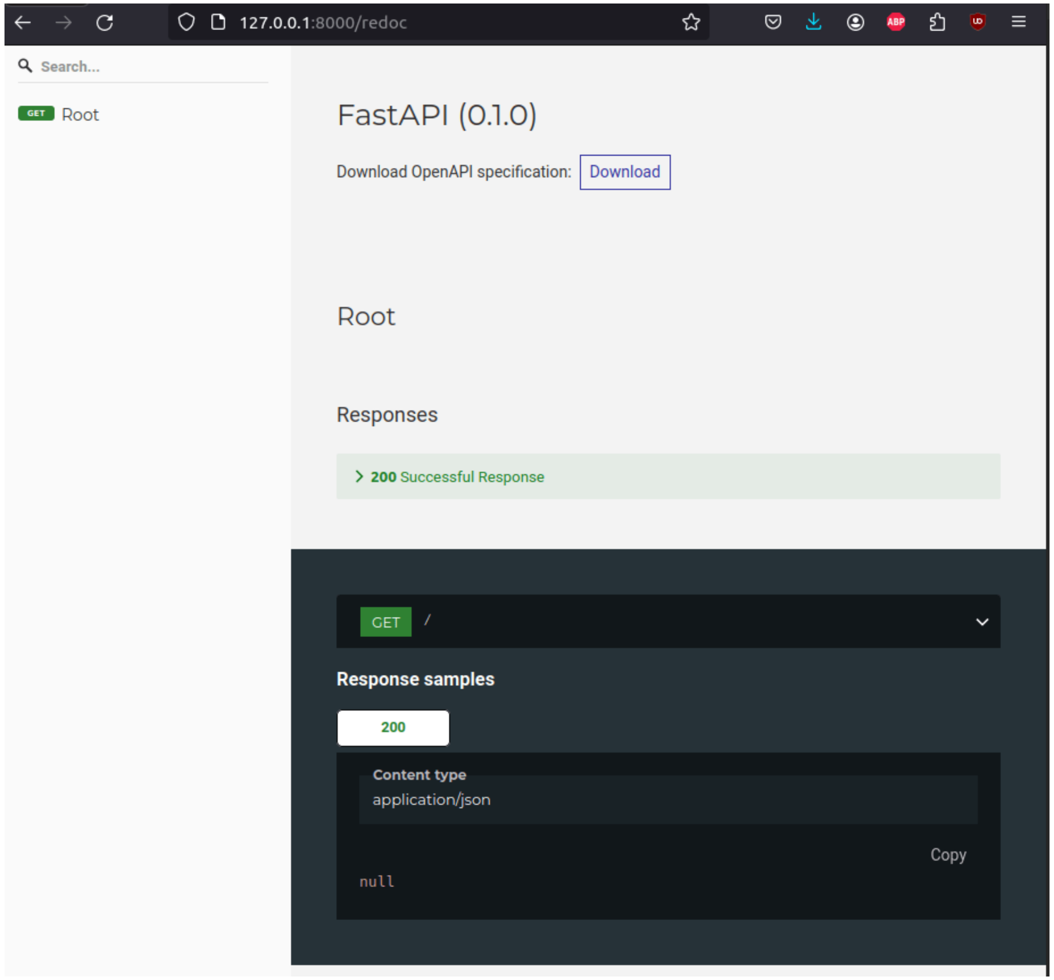
Code	Description	Links
200	Successful Response	*No links*

Media type
application/json
Controls Accept header.

Example Value | Schema

```
"string"
```

FastAPI demo - let's make the simplest API



FastAPI (0.1.0)

Download OpenAPI specification: Download

Root

Responses

200 Successful Response

GET /

Response samples

200

Content type
application/json

Copy

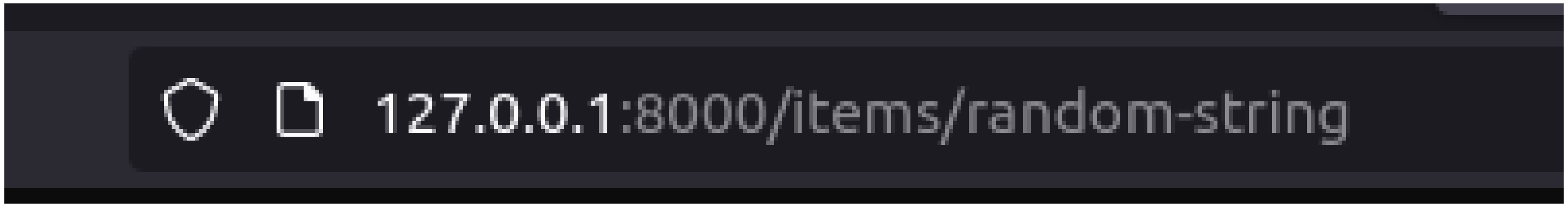
```
null
```

How about Pydantic and data validation?

How about Pydantic and data validation?

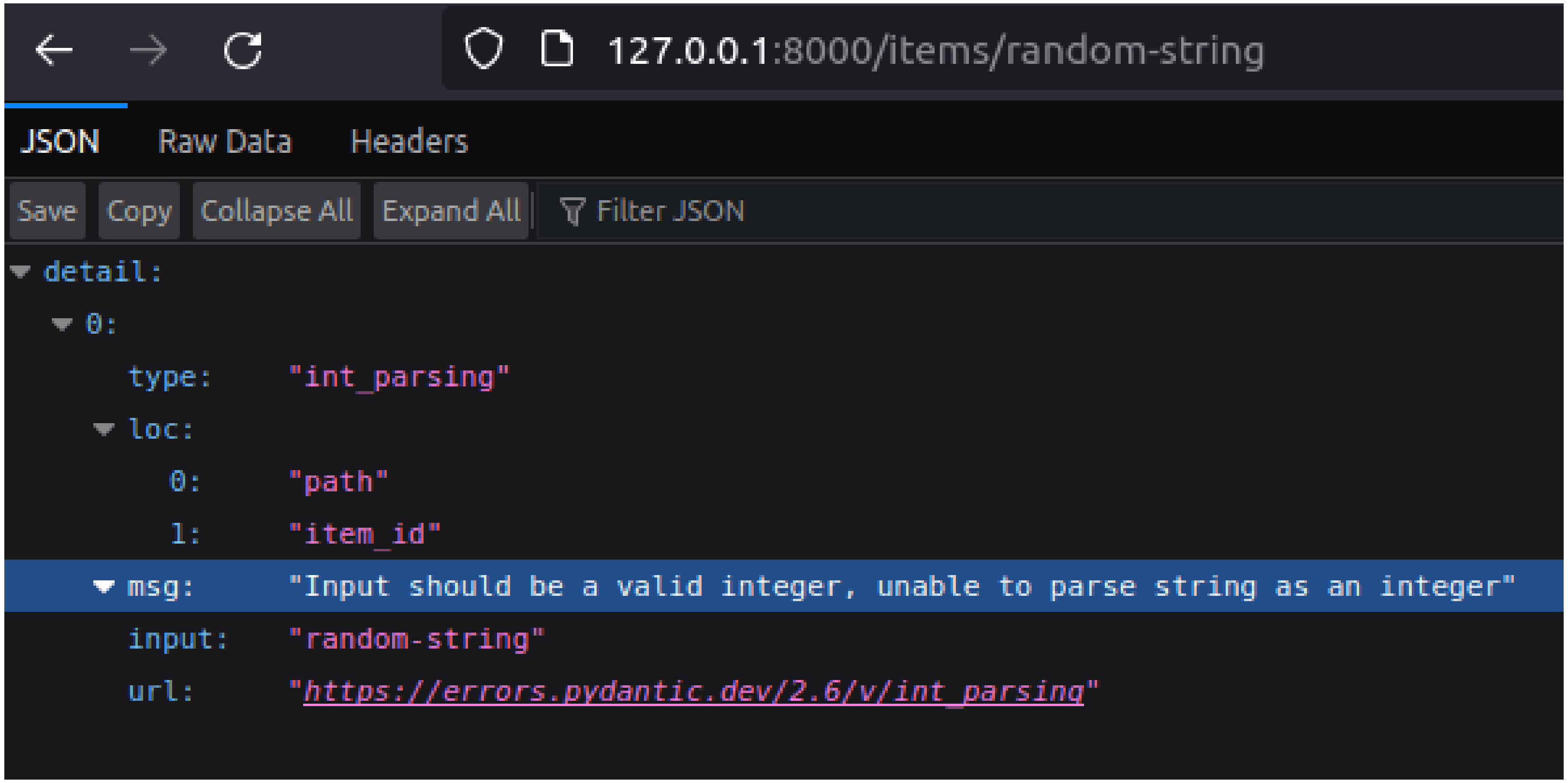
```python
@app.get("/items/{item_id}")
async def get_item(item_id: int):
    return {"item id: ", item_id}
```

How about Pydantic and data validation?



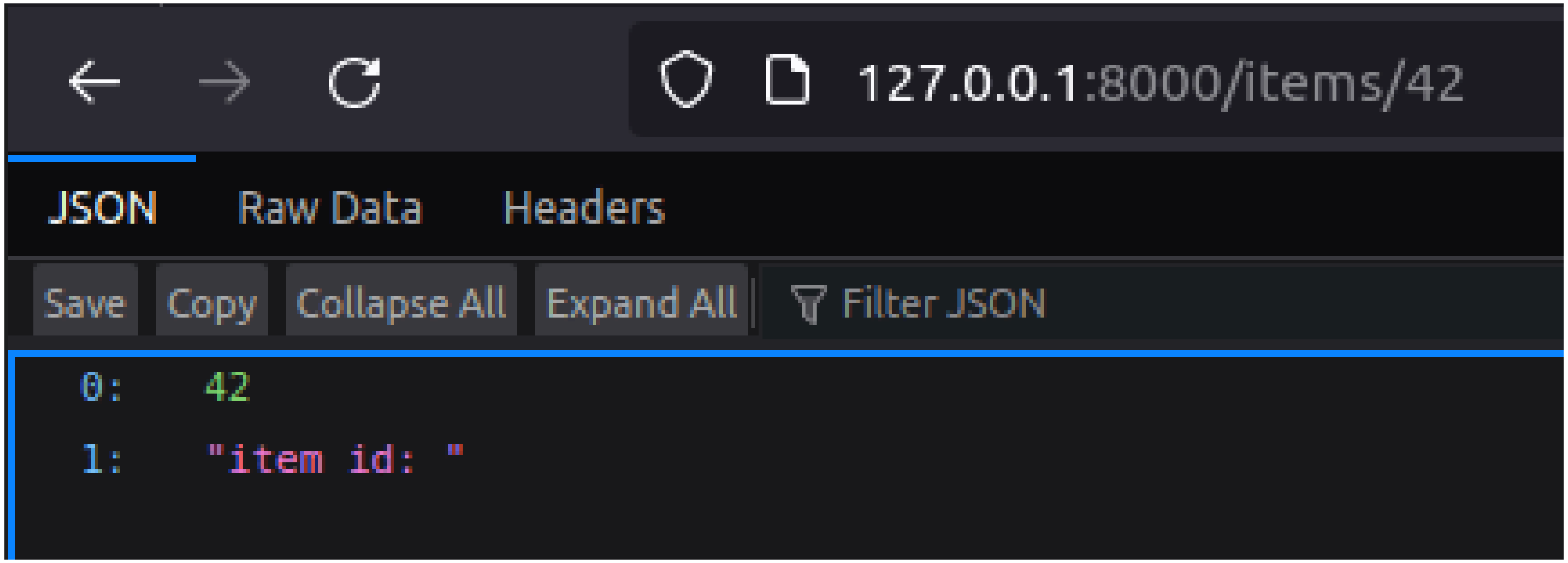
127.0.0.1:8000/items/random-string

How about Pydantic and data validation?



```
detail:
  0:
    type:     "int_parsing"
    loc:
      0:      "path"
      1:      "item_id"
    msg:      "Input should be a valid integer, unable to parse string as an integer"
    input:    "random-string"
    url:      "https://errors.pydantic.dev/2.6/v/int_parsing"
```

How about Pydantic and data validation?



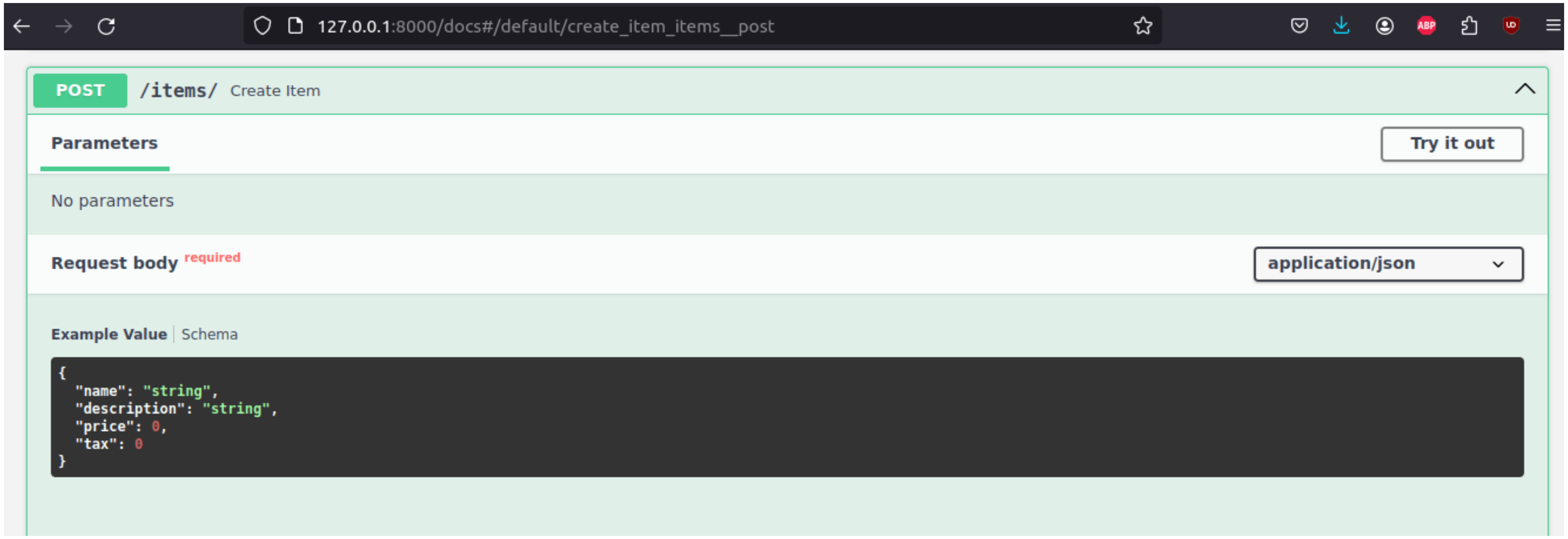
```
0:    42
1:    "item id: "
```

How about other methods?

How about other methods?

```python
class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None


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

How about other methods?


127.0.0.1:8000/docs#/default/create_item_items__post

POST /items/ Create Item

Parameters

Try it out

No parameters

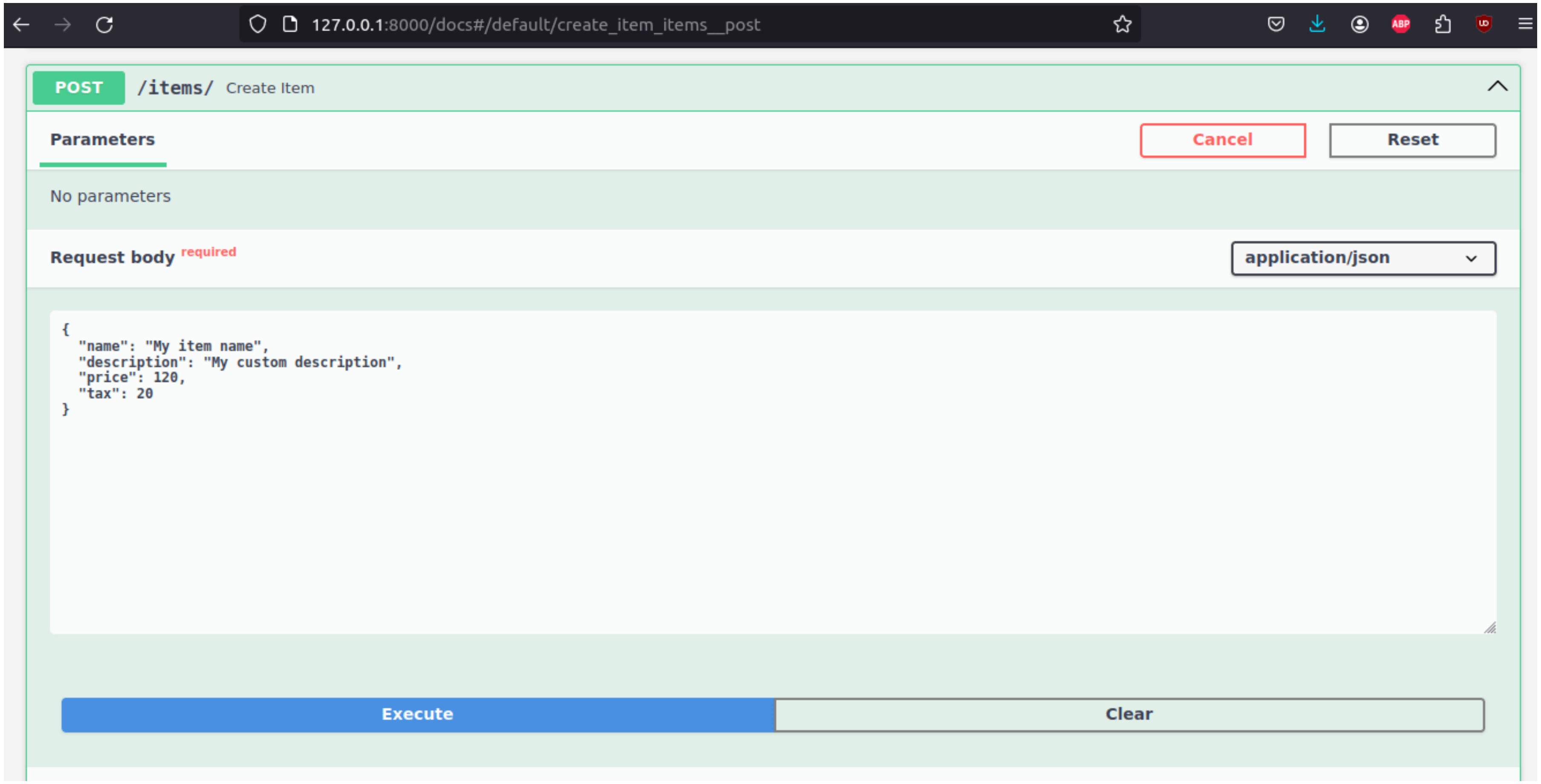
Request body required

application/json

Example Value | Schema

```
{
  "name": "string",
  "description": "string",
  "price": 0,
  "tax": 0
}
```

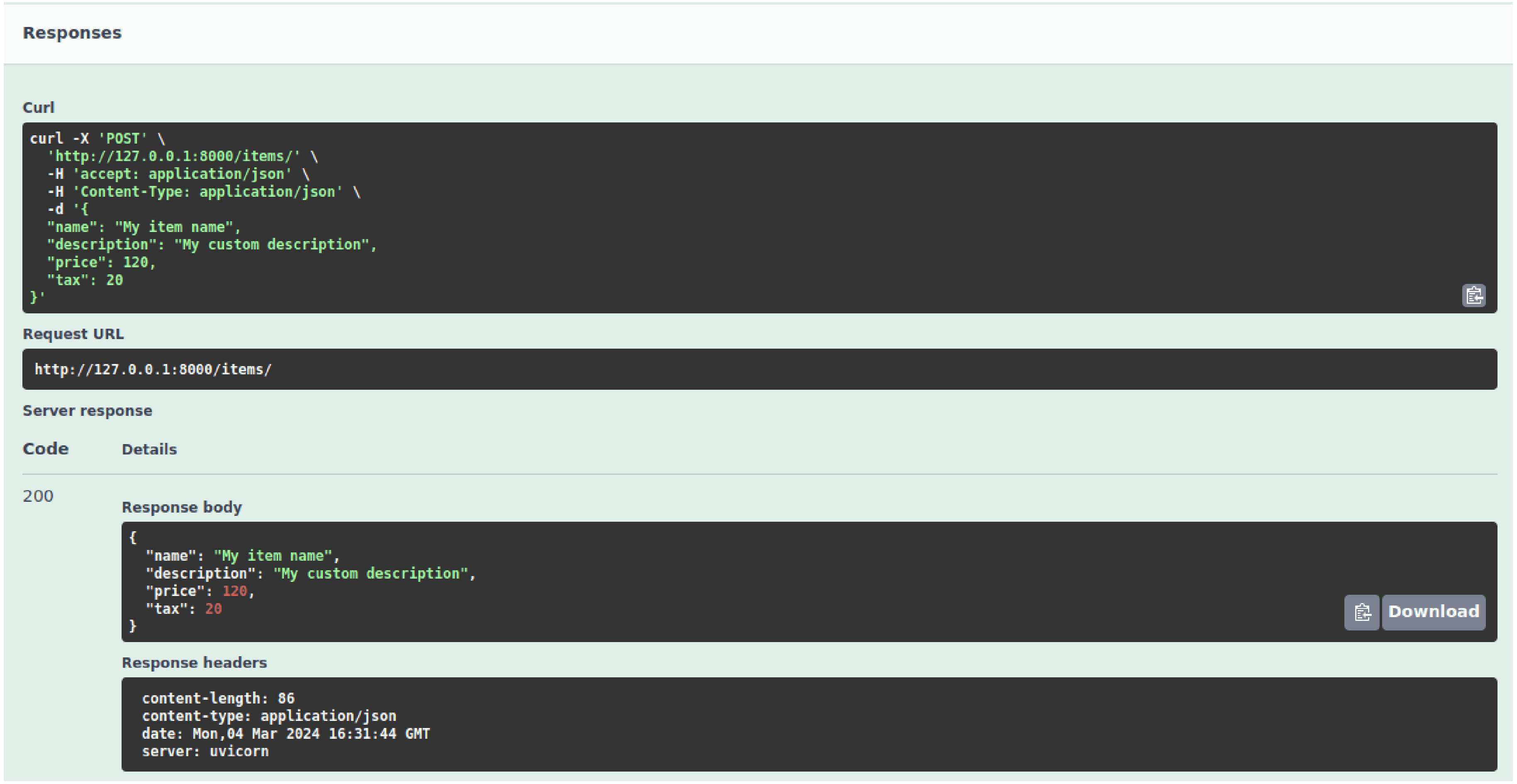
How about other methods?



No parameters

```
{
  "name": "My item name",
  "description": "My custom description",
  "price": 120,
  "tax": 20
}
```

How about other methods?



Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/items/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "name": "My item name",
  "description": "My custom description",
  "price": 120,
  "tax": 20
}'
```

Request URL

```
http://127.0.0.1:8000/items/
```

Server response

Code Details

200

Response body

```
{
  "name": "My item name",
  "description": "My custom description",
  "price": 120,
  "tax": 20
}
```

Response headers

```
 content-length: 86
 content-type: application/json
 date: Mon,04 Mar 2024 16:31:44 GMT
 server: uvicorn
```

Can I add any logic inside of my methods?

Can I add any logic inside of my methods?

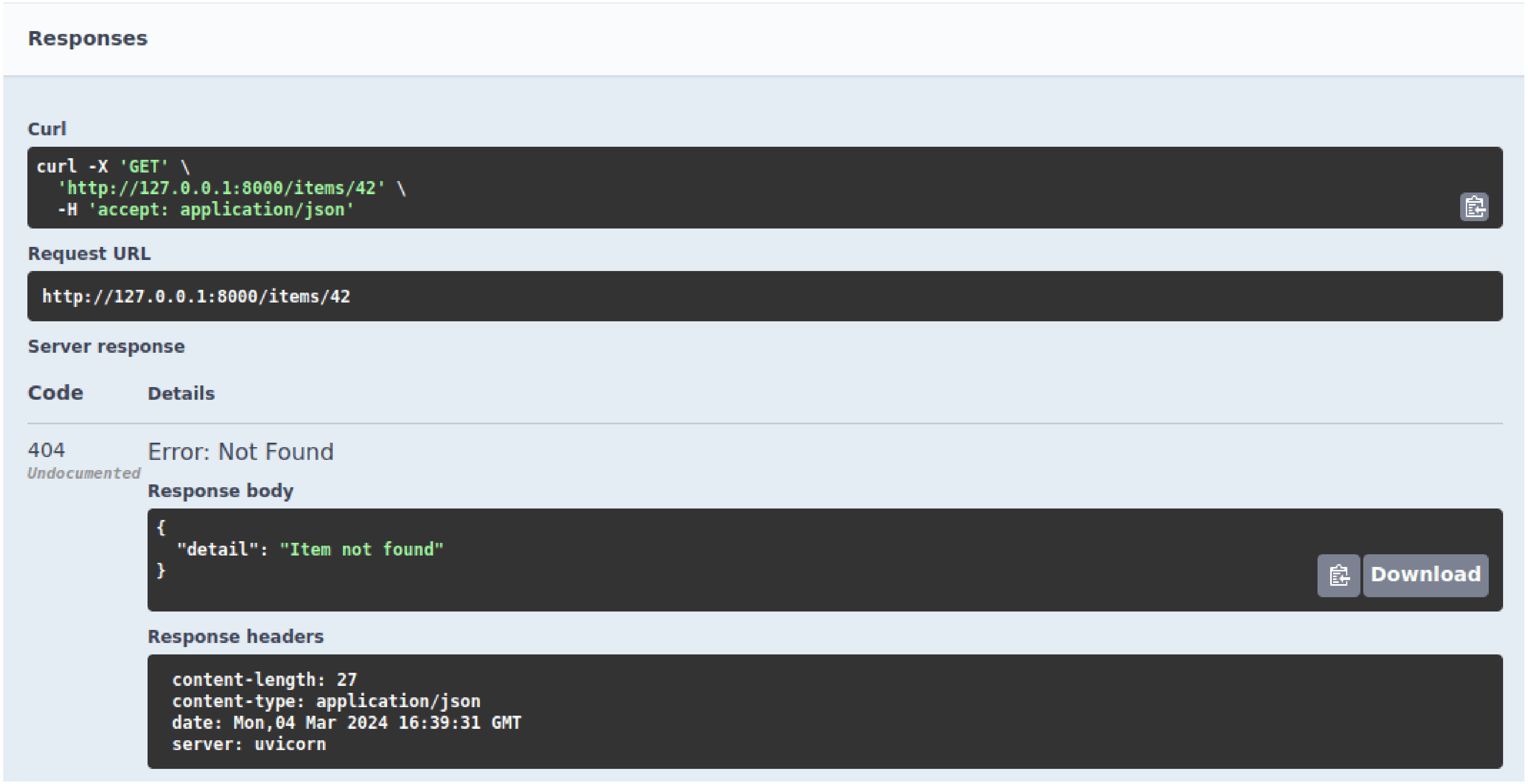
```python
@app.post("/items/")
async def create_item_with_custom_logic(item: Item):
    item_dict = item.dict()
    if item.tax:
        price_with_tax = item.price + item.tax
        item_dict.update({"price_with_tax": price_with_tax})
    return item_dict
```

What if my item doesn't exist?

What if my item doesn't exist?

```python
@app.get("/items/{item_id}")
async def read_item(item_id: str):
    if item_id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return {"item": items[item_id]}
```

What if my item doesn't exist?



Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/items/42' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/items/42
```

Server response

Code	Details
404 *Undocumented*	Error: Not Found

Response body

```
{
  "detail": "Item not found"
}
```

Response headers

```
content-length: 27
content-type: application/json
date: Mon,04 Mar 2024 16:39:31 GMT
server: uvicorn
```

Dependency Injection

Dependency Injection

```python
items = {"foo": "The Foo Wrestlers"}

def get_item(item_id: str):
    if item_id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return {"item": items[item_id]}

@app.get("/items/{item_id}")
async def read_item(item: dict = Depends(get_item)):
    return item
```

Dependency Injection - Testing Without Dependency Injection

Dependency Injection - Testing Without Dependency Injection

```python
from fastapi.testclient import TestClient
import pytest
from main import app

client = TestClient(app)

def mock_get_item(item_id: str):
    return {"item": "Mocked Item"}

@pytest.fixture
def mock_dependency(monkeypatch):
    monkeypatch.setattr("main.get_item", mock_get_item)

```

Dependency Injection - Testing With Dependency Injection

Dependency Injection - Testing With Dependency Injection

```python
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def get_item(item_id: str):
    return {"item: ": "my item"}

app.dependency_overrides[get_item] = get_item

def test_read_item():
    response = client.get("/items/foo")
    assert response.status_code == 200
```

How to structure a bigger project?

How to structure a bigger project?

```
.
├── app                  # "app" is a Python package
│   ├── __init__.py      # this file makes "app" a "Python package"
│   ├── main.py          # "main" module, e.g. import app.main
│   ├── dependencies.py  # "dependencies" module, e.g. import app.dependencies
│   └── routers          # "routers" is a "Python subpackage"
│   │   ├── __init__.py  # makes "routers" a "Python subpackage"
│   │   ├── items.py     # "items" submodule, e.g. import app.routers.items
│   │   └── users.py     # "users" submodule, e.g. import app.routers.users
│   └── internal         # "internal" is a "Python subpackage"
│       ├── __init__.py  # makes "internal" a "Python subpackage"
│       └── admin.py     # "admin" submodule, e.g. import app.internal.admin
```

Benefits of FastAPI

Benefits of FastAPI

- Exceptional Performance

Benefits of FastAPI

- Exceptional Performance
- Asynchronous Programming Support

Benefits of FastAPI

- Exceptional Performance
- Asynchronous Programming Support
- Automatic API Documentation

Benefits of FastAPI

- Exceptional Performance
- Asynchronous Programming Support
- Automatic API Documentation
- Dependency Injection
-

Benefits of FastAPI

- Exceptional Performance
- Asynchronous Programming Support
- Automatic API Documentation
- Dependency Injection
- Data Validation with Pydantic

Benefits of FastAPI

- Exceptional Performance
- Asynchronous Programming Support
- Automatic API Documentation
- Dependency Injection
- Data Validation with Pydantic
- Growing Ecosystem

Considerations for not using FastAPI

Considerations for not using FastAPI

- Smaller community and less resources

Considerations for not using FastAPI

- Smaller community and less resources
- Not ideal for CPU-bound tasks, rather for I/O-bound ones

Considerations for not using FastAPI

- Smaller community and less resources
- Not ideal for CPU-bound tasks, rather for I/O-bound ones
- Necessary to set up everything on your own - not batteries included kind of framework like Django

Considerations for not using FastAPI

- Smaller community and less resources
- Not ideal for CPU-bound tasks, rather for I/O-bound ones
- Necessary to set up everything on your own - not batteries included kind of framework like Django
- Limited project templates

Considerations for not using FastAPI

- Smaller community and less resources
- Not ideal for CPU-bound tasks, rather for I/O-bound ones
- Necessary to set up everything on your own - not batteries included kind of framework like Django
- Limited project templates
- Learning curve is steeper than in Flask, which might be more suitable for educational purposes

Resources

- awesome-fastapi repository: https://github.com/mjhea0/awesome-fastapi

Resources

- awesome-fastapi repository: https://github.com/mjhea0/awesome-fastapi
- Example real-world project written in FastAPI by Netflix: https://github.com/Netflix/dispatch

Resources

- awesome-fastapi repository: https://github.com/mjhea0/awesome-fastapi
- Example real-world project written in FastAPI by Netflix: https://github.com/Netflix/dispatch
- Best practices: https://github.com/zhanymkanov/fastapi-best-practices

Thank you!

slides

contact me