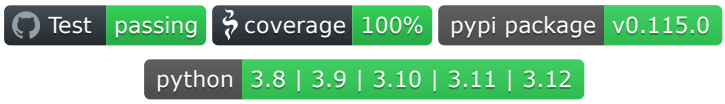
*FastAPI framework, high performance, easy to learn, fast to code, ready for production*




---

**Documentation**: https://fastapi.tiangolo.com

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

---

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 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). One of the fastest Python frameworks available.
- **Fast to code**: Increase the speed to develop features by about 200% to 300%. *
- **Fewer bugs**: Reduce about 40% of human (developer) induced errors. *
- **Intuitive**: Great editor support. Completion everywhere. Less time debugging.
- **Easy**: Designed to be easy to use and learn. Less time reading docs.
- **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- **Robust**: Get production-ready code. With automatic interactive documentation.
- **Standards-based**: Based on (and fully compatible with) the open standards for APIs: OpenAPI [↪] (previously known as Swagger) and JSON Schema [↪].

* estimation based on tests on an internal development team, building production applications.

## Sponsors

Other sponsors [↪]

## Opinions

"*[...] I'm using **FastAPI** a ton these days. [...] I'm actually planning to use it for all of my team's **ML services at Microsoft**. Some of them are getting integrated into the core **Windows** product and some **Office** products.*"

Kabir Khan - **Microsoft** (ref)

---

"*We adopted the **FastAPI** library to spawn a **REST** server that can be queried to obtain **predictions**. [for Ludwig]*"

Piero Molino, Yaroslav Dudin, and Sai Sumanth Miryala - **Uber** (ref)

---

"***Netflix** is pleased to announce the open-source release of our **crisis management** orchestration framework: **Dispatch**! [built with **FastAPI**]*"

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

---

"*I'm over the moon excited about **FastAPI**. It's so fun!*"

Brian Okken - **Python Bytes podcast host** (ref)

---

"*Honestly, what you've built looks super solid and polished. In many ways, it's what I wanted **Hug** to be - it's really inspiring to see someone build that.*"

Timothy Crosley - **Hug creator** (ref)

---

"*If you're looking to learn one **modern framework** for building REST APIs, check out **FastAPI** [...] It's fast, easy to use and easy to learn [...]*"

"*We've switched over to **FastAPI** for our **APIs** [...] I think you'll like it [...]*"

Ines Montani - Matthew Honnibal - **Explosion AI founders - spaCy creators** (ref) - (ref)

*"If anyone is looking to build a production Python API, I would highly recommend **FastAPI**. It is **beautifully designed**, **simple to use** and **highly scalable**, it has become a **key component** in our API first development strategy and is driving many automations and services such as our Virtual TAC Engineer."*

Deon Pillsbury - **Cisco** (ref)

## **Typer**, the FastAPI of CLIs


If you are building a CLI app to be used in the terminal instead of a web API, check out **Typer** [↪].

**Typer** is FastAPI's little sibling. And it's intended to be the **FastAPI of CLIs**. ⌨ 🚀

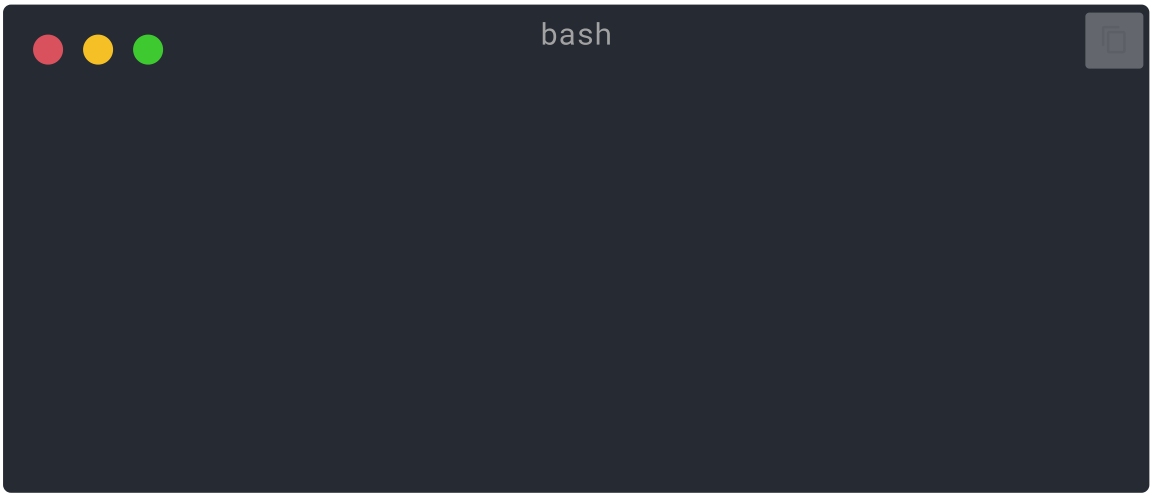
## Requirements

FastAPI stands on the shoulders of giants:

- Starlette [↪] for the web parts.
- Pydantic [↪] for the data parts.

## Installation

Create and activate a virtual environment [↪] and then install FastAPI:



**Note**: Make sure you put `"fastapi[standard]"` in quotes to ensure it works in all terminals.

## Example

### Create it

- Create a file `main.py` with:

```
from typing import Union

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

**Or use `async def`...**

If your code uses `async` / `await`, use `async def`:

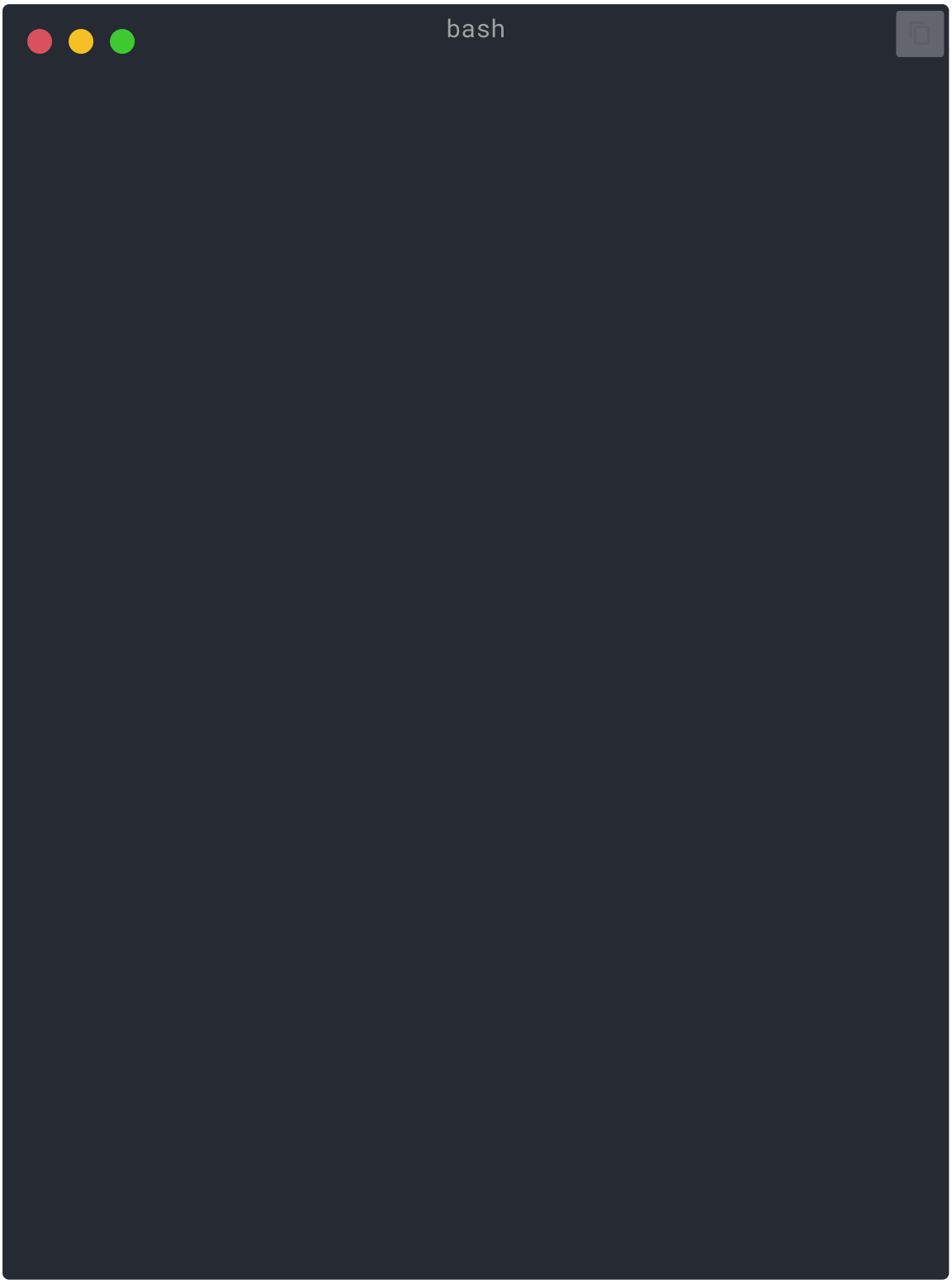
```
from typing import Union

from fastapi import FastAPI

app = FastAPI()


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


@app.get("/items/{item_id}")
async def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

**Note**:

If you don't know, check the *"In a hurry?"* section about `async` and `await` in the docs.

### Run it

Run the server with:



**About the command `fastapi dev main.py`...**

The command `fastapi dev` reads your `main.py` file, detects the **FastAPI** app in it, and starts a server using Uvicorn [↪].

By default, `fastapi dev` will start with auto-reload enabled for local development.

You can read more about it in the FastAPI CLI docs.

## Check it

Open your browser at http://127.0.0.1:8000/items/5?q=somequery [↪].

You will see the JSON response as:

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

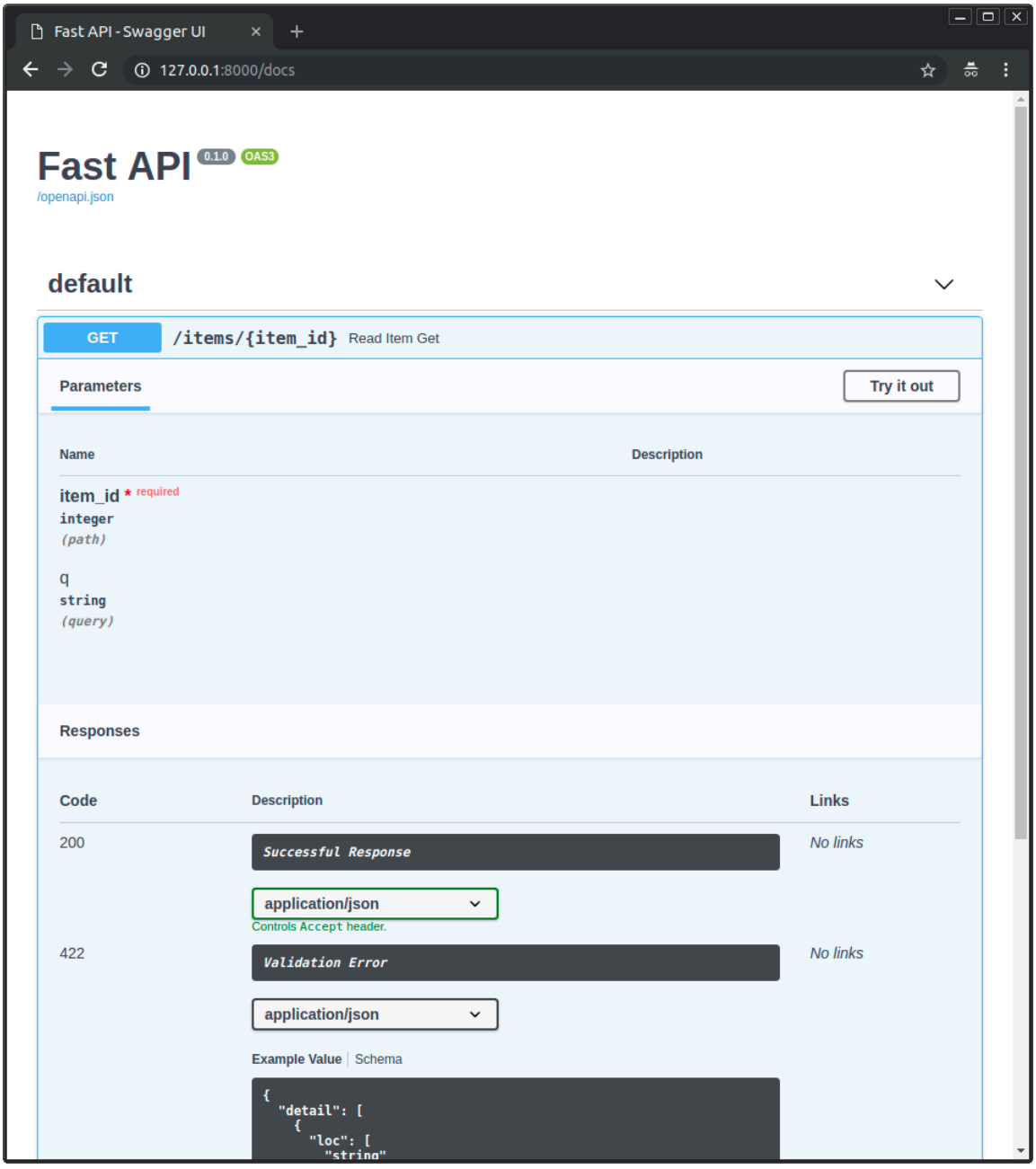
You already created an API that:

- Receives HTTP requests in the *paths* `/` and `/items/{item_id}`.
- Both *paths* take `GET` *operations* (also known as HTTP *methods*).
- The *path* `/items/{item_id}` has a *path parameter* `item_id` that should be an `int`.
- The *path* `/items/{item_id}` has an optional `str` *query parameter* `q`.

## Interactive API docs

Now go to http://127.0.0.1:8000/docs [↪].

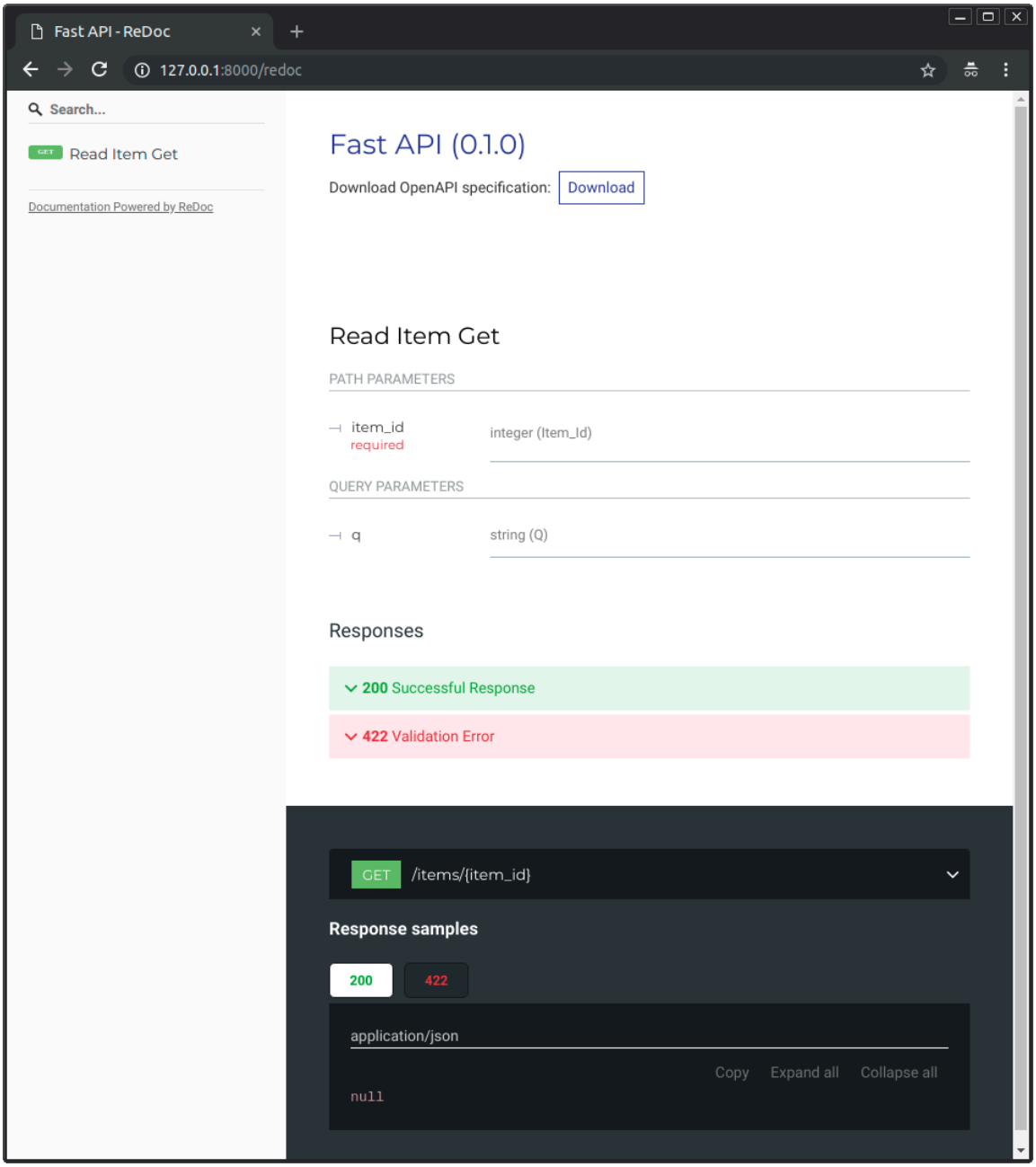
You will see the automatic interactive API documentation (provided by Swagger UI [↪]):




## Alternative API docs

And now, go to http://127.0.0.1:8000/redoc [↪].

You will see the alternative automatic documentation (provided by ReDoc [↪]):




## Example upgrade

Now modify the file `main.py` to receive a body from a `PUT` request.

Declare the body using standard Python types, thanks to Pydantic.

```
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()
```

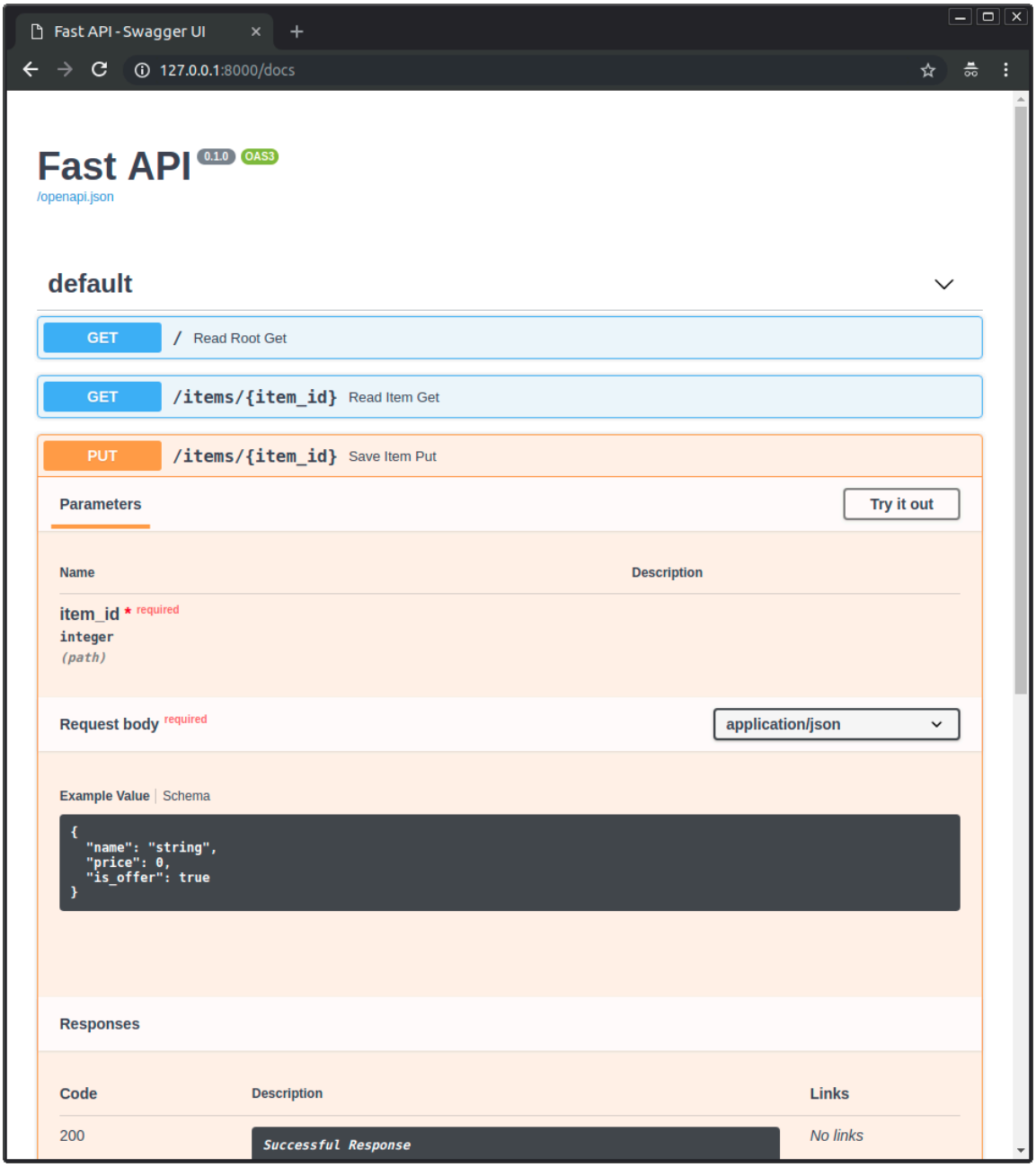
```python
class Item(BaseModel):
    name: str
    price: float
    is_offer: Union[bool, None] = None


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


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = 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}
```

The `fastapi dev` server should reload automatically.

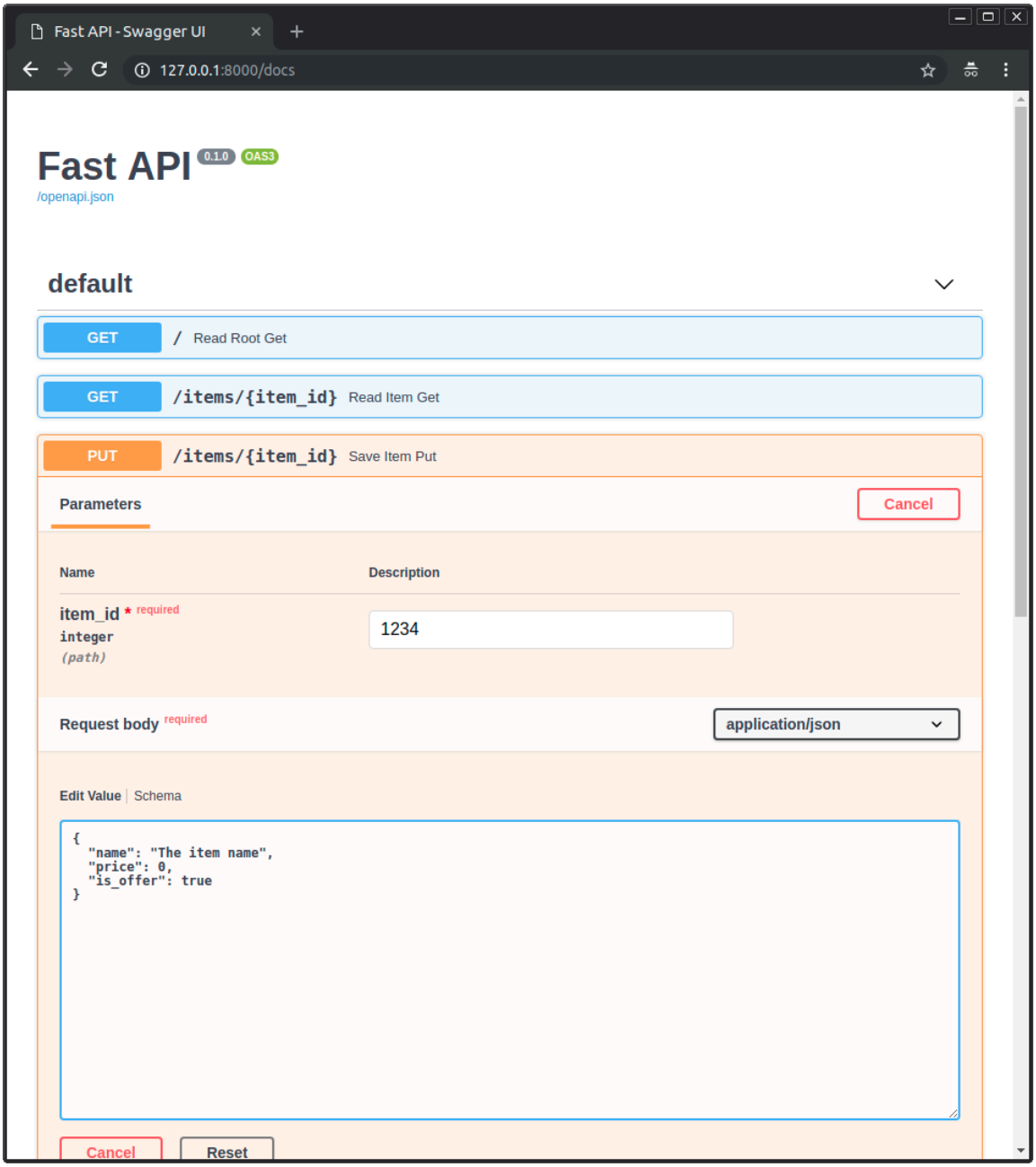
## Interactive API docs upgrade

Now go to http://127.0.0.1:8000/docs [↪].

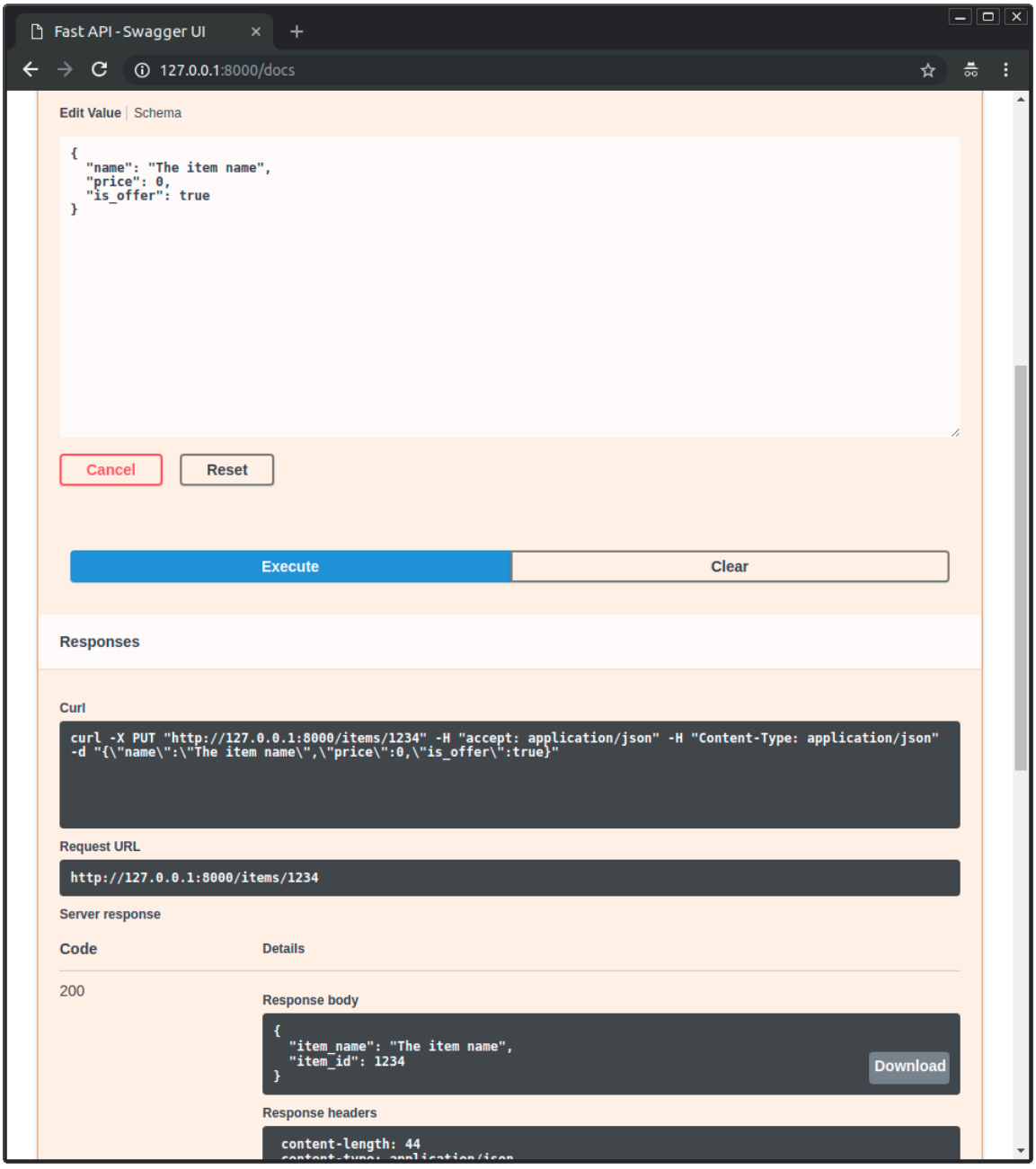
- The interactive API documentation will be automatically updated, including the new body:




- Click on the button "Try it out", it allows you to fill the parameters and directly interact with the API:



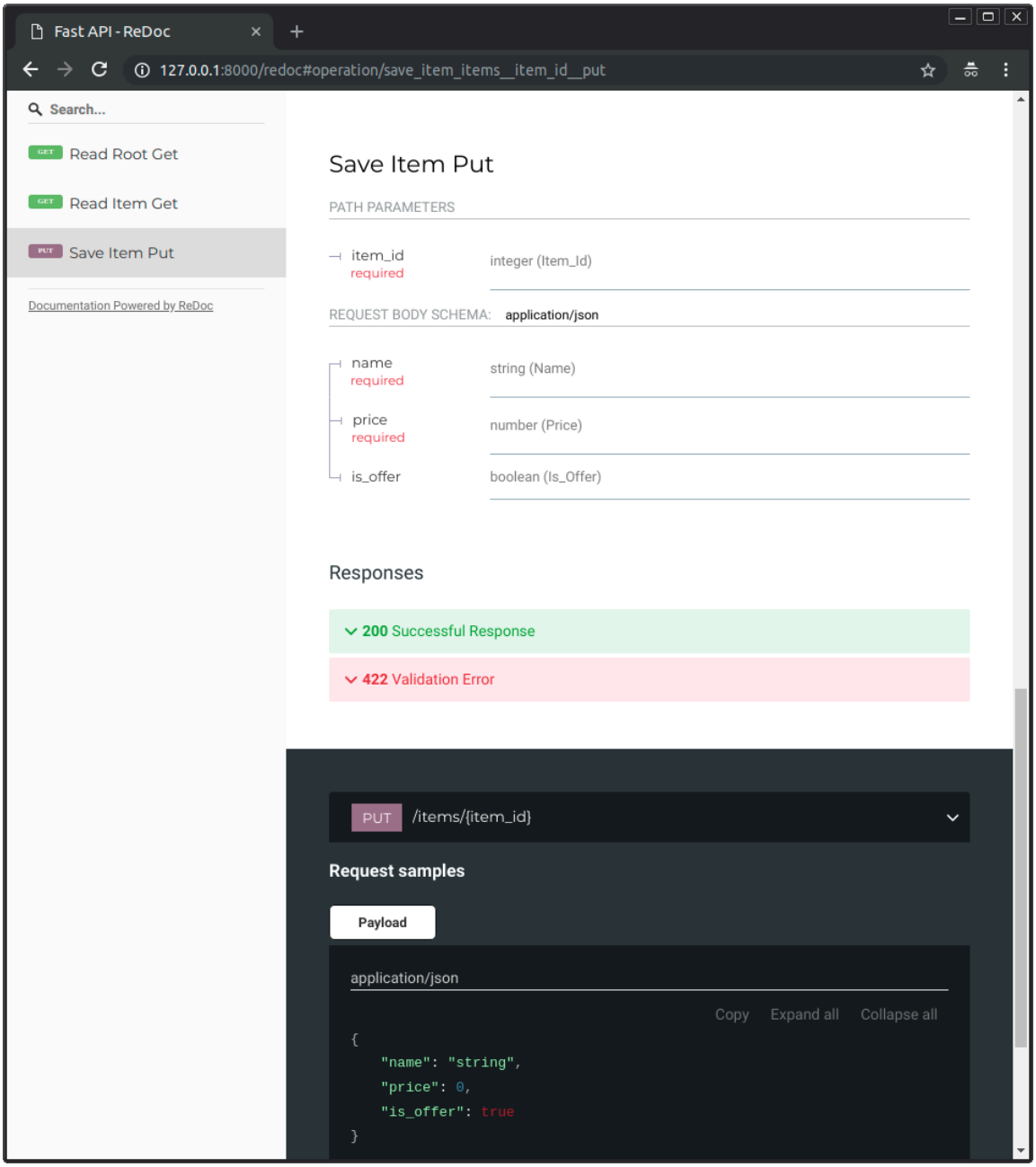

- Then click on the "Execute" button, the user interface will communicate with your API, send the parameters, get the results and show them on the screen:




## Alternative API docs upgrade

And now, go to http://127.0.0.1:8000/redoc [↪].

- The alternative documentation will also reflect the new query parameter and body:




## Recap

In summary, you declare **once** the types of parameters, body, etc. as function parameters.

You do that with standard modern Python types.

You don't have to learn a new syntax, the methods or classes of a specific library, etc.

Just standard **Python**.

For example, for an `int`:

```
item_id: int
```

or for a more complex `Item` model:

```
item: Item
```

...and with that single declaration you get:

- Editor support, including:
  - Completion.
  - Type checks.
- Validation of data:
  - Automatic and clear errors when the data is invalid.
  - Validation even for deeply nested JSON objects.
- Conversion of input data: coming from the network to Python data and types. Reading from:
  - JSON.
  - Path parameters.
  - Query parameters.
  - Cookies.
  - Headers.
  - Forms.
  - Files.
- Conversion of output data: converting from Python data and types to network data (as JSON):
  - Convert Python types (`str`, `int`, `float`, `bool`, `list`, etc).
  - `datetime` objects.
  - `UUID` objects.
  - Database models.
  - ...and many more.
- Automatic interactive API documentation, including 2 alternative user interfaces:
  - Swagger UI.
  - ReDoc.

---

Coming back to the previous code example, **FastAPI** will:

- Validate that there is an `item_id` in the path for `GET` and `PUT` requests.

- Validate that the `item_id` is of type `int` for `GET` and `PUT` requests.
  - If it is not, the client will see a useful, clear error.
- Check if there is an optional query parameter named `q` (as in `http://127.0.0.1:8000/items/foo?q=somequery`) for `GET` requests.
  - As the `q` parameter is declared with `= None`, it is optional.
  - Without the `None` it would be required (as is the body in the case with `PUT`).
- For `PUT` requests to `/items/{item_id}`, read the body as JSON:
  - Check that it has a required attribute `name` that should be a `str`.
  - Check that it has a required attribute `price` that has to be a `float`.
  - Check that it has an optional attribute `is_offer`, that should be a `bool`, if present.
  - All this would also work for deeply nested JSON objects.
- Convert from and to JSON automatically.
- Document everything with OpenAPI, that can be used by:
  - Interactive documentation systems.
  - Automatic client code generation systems, for many languages.
- Provide 2 interactive documentation web interfaces directly.

---

We just scratched the surface, but you already get the idea of how it all works.

Try changing the line with:

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

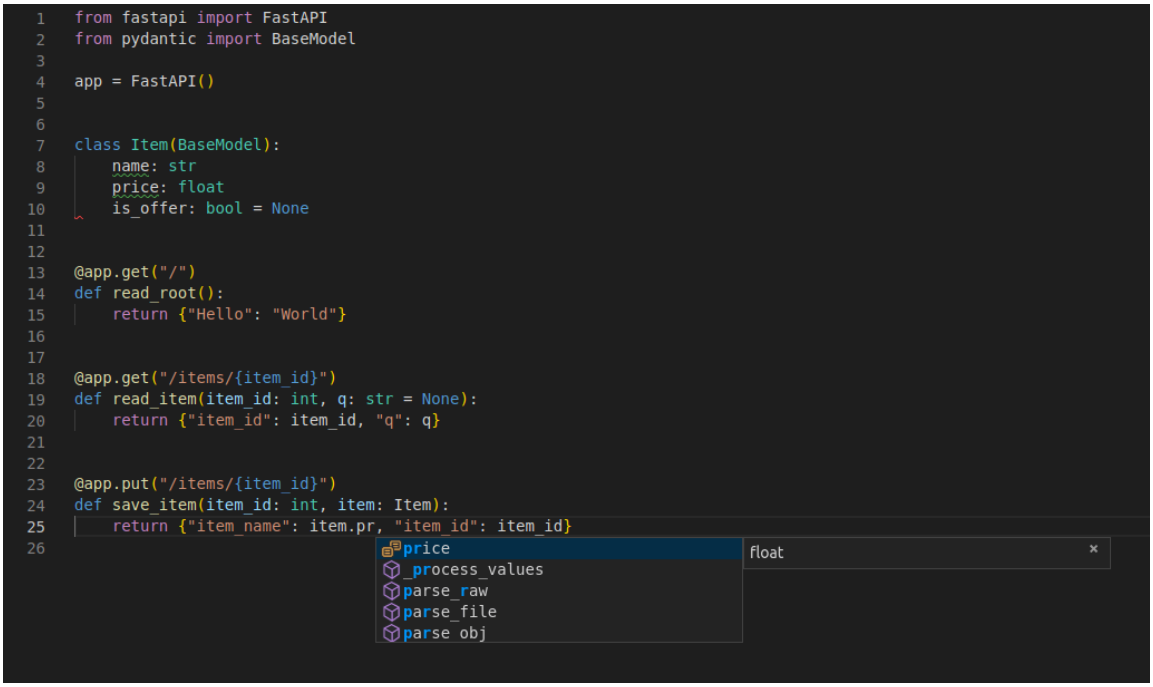
...from:

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

...to:

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

...and see how your editor will auto-complete the attributes and know their types:




For a more complete example including more features, see the Tutorial - User Guide.

**Spoiler alert**: the tutorial - user guide includes:

- Declaration of **parameters** from other different places as: **headers**, **cookies**, **form fields** and **files**.
- How to set **validation constraints** as `maximum_length` or `regex`.
- A very powerful and easy to use **Dependency Injection** system.
- Security and authentication, including support for **OAuth2** with **JWT tokens** and **HTTP Basic** auth.
- More advanced (but equally easy) techniques for declaring **deeply nested JSON models** (thanks to Pydantic).
- **GraphQL** integration with Strawberry [↪] and other libraries.
- Many extra features (thanks to Starlette) as:
  - **WebSockets**
  - extremely easy tests based on HTTPX and `pytest`
  - **CORS**
  - **Cookie Sessions**
  - ...and more.

## Performance

Independent TechEmpower benchmarks show **FastAPI** applications running under Uvicorn as one of the fastest Python frameworks available [↪], only below Starlette and Uvicorn themselves (used internally by FastAPI). (*)

To understand more about it, see the section Benchmarks ↪.

## Dependencies

FastAPI depends on Pydantic and Starlette.

### `standard` Dependencies

When you install FastAPI with `pip install "fastapi[standard]"` it comes the `standard` group of optional dependencies:

Used by Pydantic:

- `email-validator` - for email validation.

Used by Starlette:

- `httpx` - Required if you want to use the `TestClient`.
- `jinja2` - Required if you want to use the default template configuration.
- `python-multipart` - Required if you want to support form "parsing", with `request.form()`.

Used by FastAPI / Starlette:

- `uvicorn` - for the server that loads and serves your application. This includes `uvicorn[standard]`, which includes some dependencies (e.g. `uvloop`) needed for high performance serving.
- `fastapi-cli` - to provide the `fastapi` command.

### Without `standard` Dependencies

If you don't want to include the `standard` optional dependencies, you can install with `pip install fastapi` instead of `pip install "fastapi[standard]"`.

### Additional Optional Dependencies

There are some additional dependencies you might want to install.

Additional optional Pydantic dependencies:

- `pydantic-settings` - for settings management.
- `pydantic-extra-types` - for extra types to be used with Pydantic.

Additional optional FastAPI dependencies:

- `orjson` - Required if you want to use `ORJSONResponse`.
- `ujson` - Required if you want to use `UJSONResponse`.

## License

This project is licensed under the terms of the MIT license.

**Was this page helpful?**