First Steps

The simplest FastAPI file could look like this:

```python
from fastapi import FastAPI

app = FastAPI()


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

Copy that to a file `main.py`.

Run the live server:

```
bash

$ 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.

                                      restart ↻
```

Note

The command `uvicorn main:app` refers to:

- `main`: the file `main.py` (the Python "module").
- `app`: the object created inside of `main.py` with the line `app = FastAPI()`.
- `--reload`: make the server restart after code changes. Only use for development.

In the output, there's a line with something like:

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

That line shows the URL where your app is being served, in your local machine.

Check it

Open your browser at http://127.0.0.1:8000 [↪].

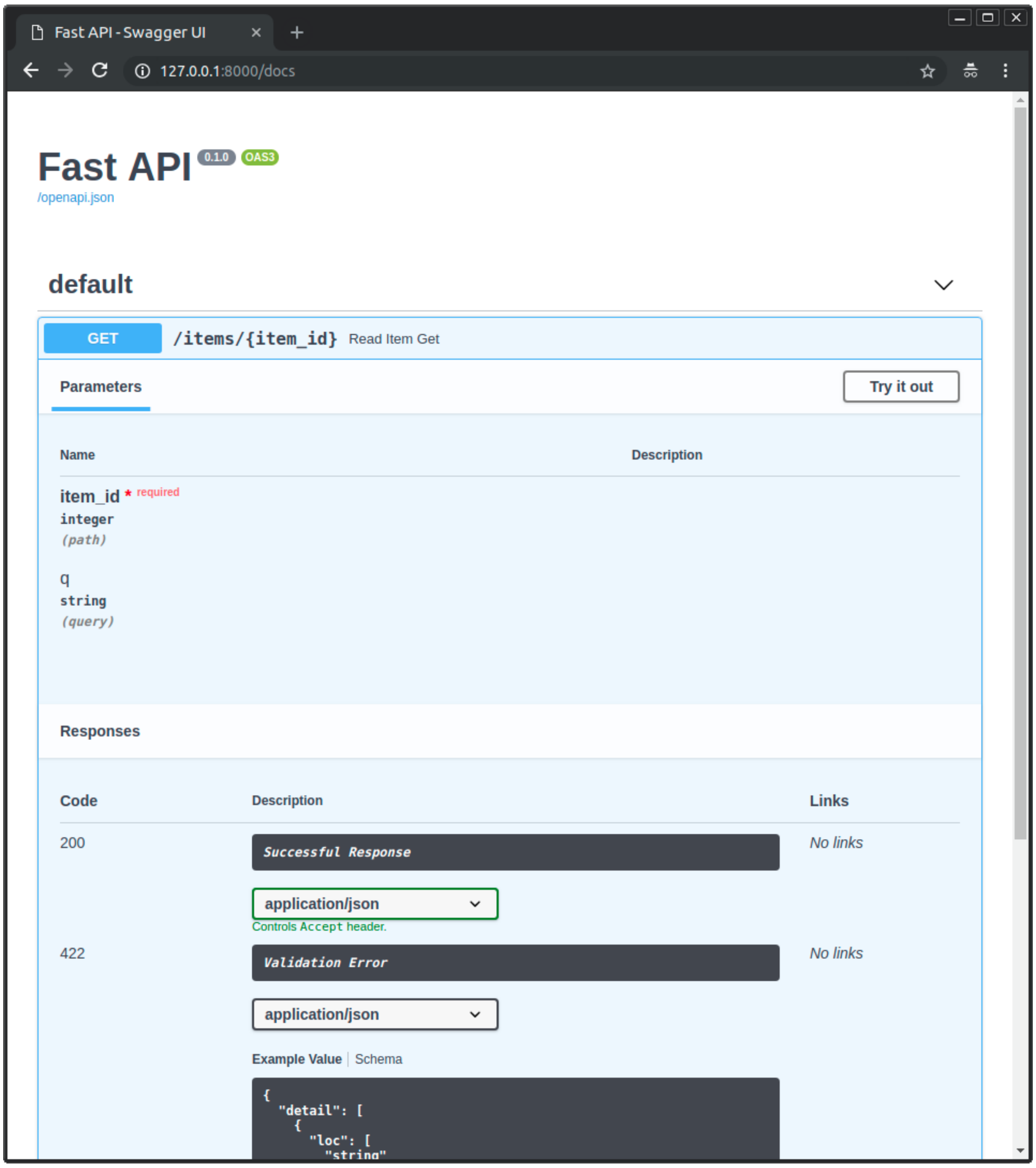
You will see the JSON response as:

```
{"message": "Hello World"}
```

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 [↪]):



Fast API 0.1.0 OAS3
/openapi.json

default

GET /items/{item_id} Read Item Get

Parameters Try it out

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

Responses

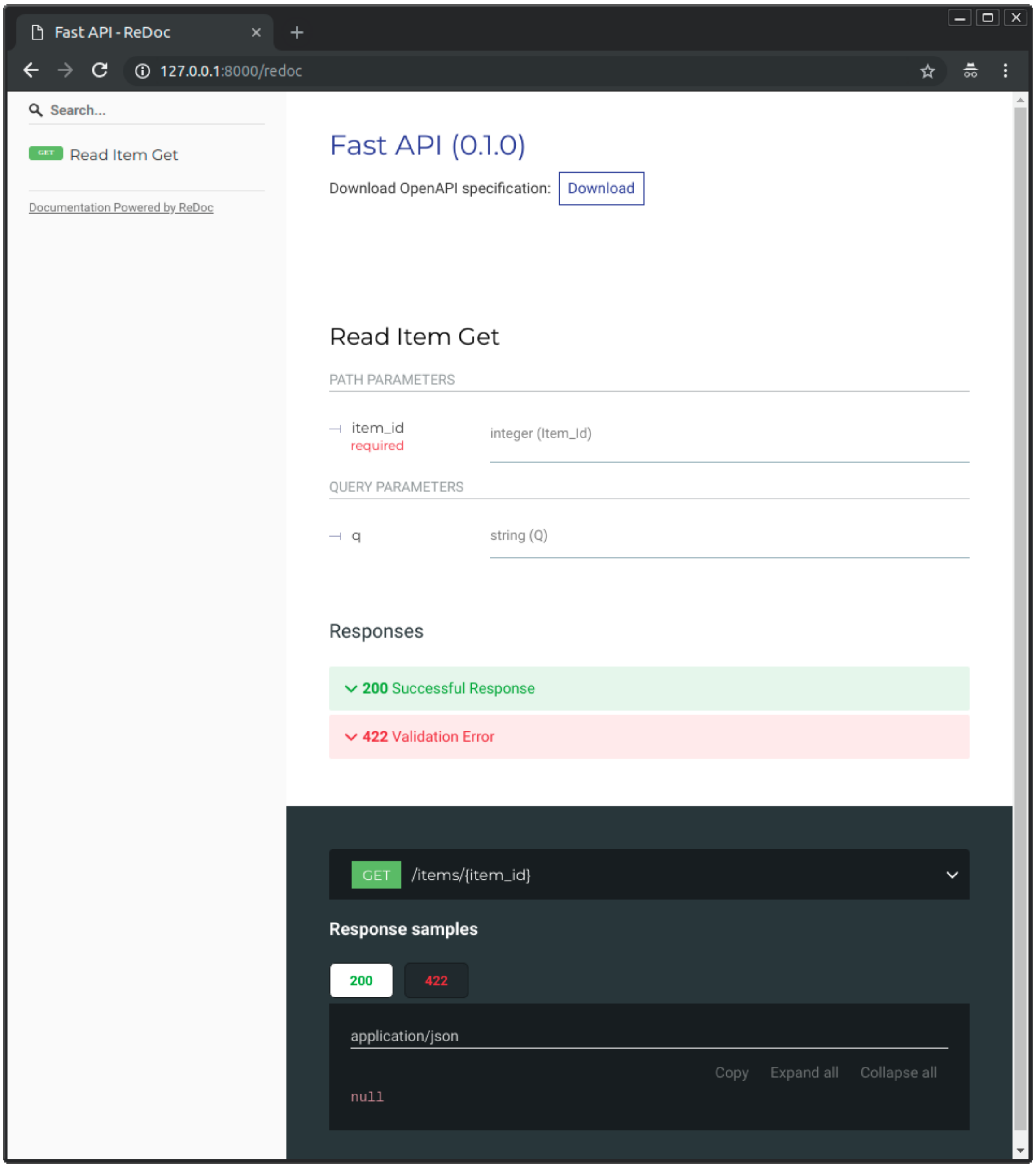
Code	Description	Links
200	*Successful Response* application/json Controls Accept header.	*No links*
422	*Validation Error* application/json Example Value | Schema	*No links*

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

Alternative API docs

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

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



Fast API (0.1.0)

Download OpenAPI specification: Download

Read Item Get

PATH PARAMETERS

item_id required	integer (Item_Id)

QUERY PARAMETERS

q	string (Q)

Responses

200 Successful Response

422 Validation Error

GET /items/{item_id}

Response samples

200 422

application/json

Copy Expand all Collapse all

```
null
```

OpenAPI

FastAPI generates a "schema" with all your API using the **OpenAPI** standard for defining APIs.

"Schema"

A "schema" is a definition or description of something. Not the code that implements it, but just an abstract description.

API "schema"

In this case, OpenAPI [↪] is a specification that dictates how to define a schema of yo

This schema definition includes your API paths, the possible parameters they take, etc.

Data "schema"

The term "schema" might also refer to the shape of some data, like a JSON content.

In that case, it would mean the JSON attributes, and data types they have, etc.

OpenAPI and JSON Schema

OpenAPI defines an API schema for your API. And that schema includes definitions (or "schemas") of the data sent and received by your API using **JSON Schema**, the standard for JSON data schemas.

Check the `openapi.json`

If you are curious about how the raw OpenAPI schema looks like, FastAPI automatically generates a JSON (schema) with the descriptions of all your API.

You can see it directly at: http://127.0.0.1:8000/openapi.json [↪].

It will show a JSON starting with something like:

```
{
    "openapi": "3.0.2",
    "info": {
        "title": "FastAPI",
        "version": "0.1.0"
    },
    "paths": {
        "/items/": {
            "get": {
                "responses": {
                    "200": {
                        "description": "Successful Response",
                        "content": {
                            "application/json": {



...
```

What is OpenAPI for

The OpenAPI schema is what powers the two interactive documentation systems included.

And there are dozens of alternatives, all based on OpenAPI. You could easily add any of those alternatives to your application built with **FastAPI**.

You could also use it to generate code automatically, for clients that communicate wi
API. For example, frontend, mobile or IoT applications.

Recap, step by step

Step 1: import `FastAPI`

```python
from fastapi import FastAPI

app = FastAPI()


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

`FastAPI` is a Python class that provides all the functionality for your API.

Technical Details

`FastAPI` is a class that inherits directly from `Starlette`.

You can use all the Starlette [↪] functionality with `FastAPI` too.

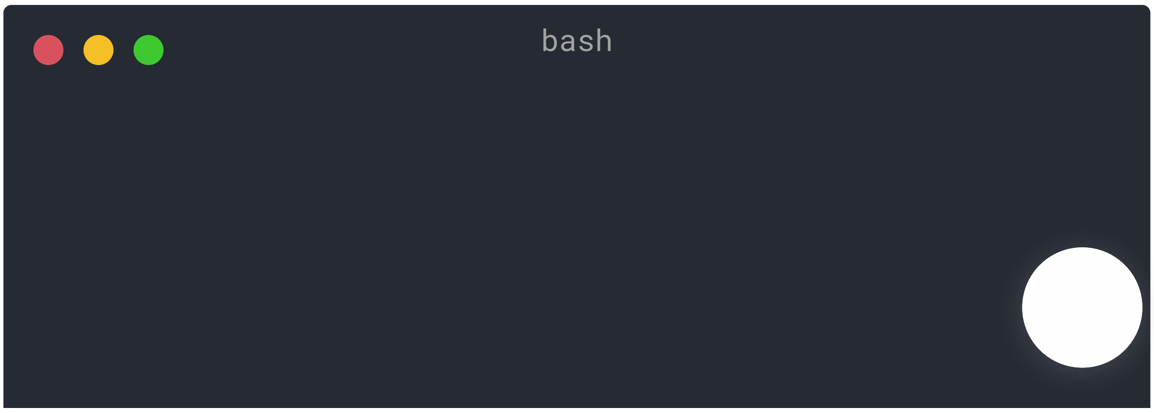
Step 2: create a `FastAPI` "instance"

```python
from fastapi import FastAPI

app = FastAPI()


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

Here the `app` variable will be an "instance" of the class `FastAPI`.

This will be the main point of interaction to create all your API.

This `app` is the same one referred by `uvicorn` in the command:



If you create your app like:

```python
from fastapi import FastAPI

my_awesome_api = FastAPI()


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

And put it in a file `main.py`, then you would call `uvicorn` like:

Step 3: create a *path operation*

Path

"Path" here refers to the last part of the URL starting from the first `/`.

So, in a URL like:

```
https://example.com/items/foo
```

...the path would be:

```
/items/foo
```

Info

A "path" is also commonly called an "endpoint" or a "route".

While building an API, the "path" is the main way to separate "concerns" and "resources".

Operation

"Operation" here refers to one of the HTTP "methods".

One of:

- `POST`
- `GET`
- `PUT`
- `DELETE`

...and the more exotic ones:

- `OPTIONS`
- `HEAD`
- `PATCH`
- `TRACE`

In the HTTP protocol, you can communicate to each path using one (or more) of these "methods".

When building APIs, you normally use these specific HTTP methods to perform a specific action.

Normally you use:

- `POST` : to create data.
- `GET` : to read data.
- `PUT` : to update data.
- `DELETE` : to delete data.

So, in OpenAPI, each of the HTTP methods is called an "operation".

We are going to call them "**operations**" too.

Define a *path operation decorator*

```python
from fastapi import FastAPI

app = FastAPI()


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

The `@app.get("/")` tells **FastAPI** that the function right below is in charge of handling requests that go to:

- the path `/`
- using a `get` operation

`@decorator` Info

That `@something` syntax in Python is called a "decorator".

You put it on top of a function. Like a pretty decorative hat (I guess that's where the term came from).

A "decorator" takes the function below and does something with it.

In our case, this decorator tells **FastAPI** that the function below corresponds to the **path** `/` with an **operation** `get`.

It is the "**path operation decorator**".

You can also use the other operations:

- `@app.post()`
- `@app.put()`
- `@app.delete()`

And the more exotic ones:

- `@app.options()`
- `@app.head()`
- `@app.patch()`
- `@app.trace()`

Tip

You are free to use each operation (HTTP method) as you wish.

FastAPI doesn't enforce any specific meaning.

The information here is presented as a guideline, not a requirement.

For example, when using GraphQL you normally perform all the actions using only `POST` operations.

Step 4: define the **path operation function**

This is our "**path operation function**":

- **path**: is `/`.
- **operation**: is `get`.
- **function**: is the function below the "decorator" (below `@app.get("/")`).

```python
from fastapi import FastAPI

app = FastAPI()


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

This is a Python function.

It will be called by **FastAPI** whenever it receives a request to the URL "`/`" using a `GET` operation.

In this case, it is an `async` function.

You could also define it as a normal function instead of `async def`:

```python
from fastapi import FastAPI

app = FastAPI()


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

Note

If you don't know the difference, check the Async: *"In a hurry?"* ↪.

Step 5: return the content

```python
from fastapi import FastAPI

app = FastAPI()


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

You can return a `dict`, `list`, singular values as `str`, `int`, etc.

You can also return Pydantic models (you'll see more about that later).

There are many other objects and models that will be automatically converted to JSON (including ORMs, etc). Try using your favorite ones, it's highly probable that they are already supported.

Recap

- Import `FastAPI`.
- Create an `app` instance.
- Write a **path operation decorator** (like `@app.get("/")`).
- Write a **path operation function** (like `def root(): ...` above).
- Run the development server (like `uvicorn main:app --reload`).