Primeros pasos

Un archivo muy simple de FastAPI podría verse así:

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Copia eso a un archivo `main.py`.

Corre el servidor en vivo:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
<span style="color: green;">INFO</span>:     Started reloader process [28720]
<span style="color: green;">INFO</span>:     Started server process [28722]
<span style="color: green;">INFO</span>:     Waiting for application startup.
<span style="color: green;">INFO</span>:     Application startup complete.
```

!!! note "Nota" El comando `uvicorn main:app` se refiere a:

```
* `main`: el archivo `main.py` (el "módulo" de Python).
* `app`: el objeto creado dentro de `main.py` con la línea `app = FastAPI()`.
* `--reload`: hace que el servidor se reinicie cada vez que cambia el código. Úsalo
únicamente para desarrollo.
```

En el output, hay una línea que dice más o menos:

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

Esa línea muestra la URL dónde se está sirviendo tu app en tu maquina local.

Revísalo

Abre tu navegador en http://127.0.0.1:8000.

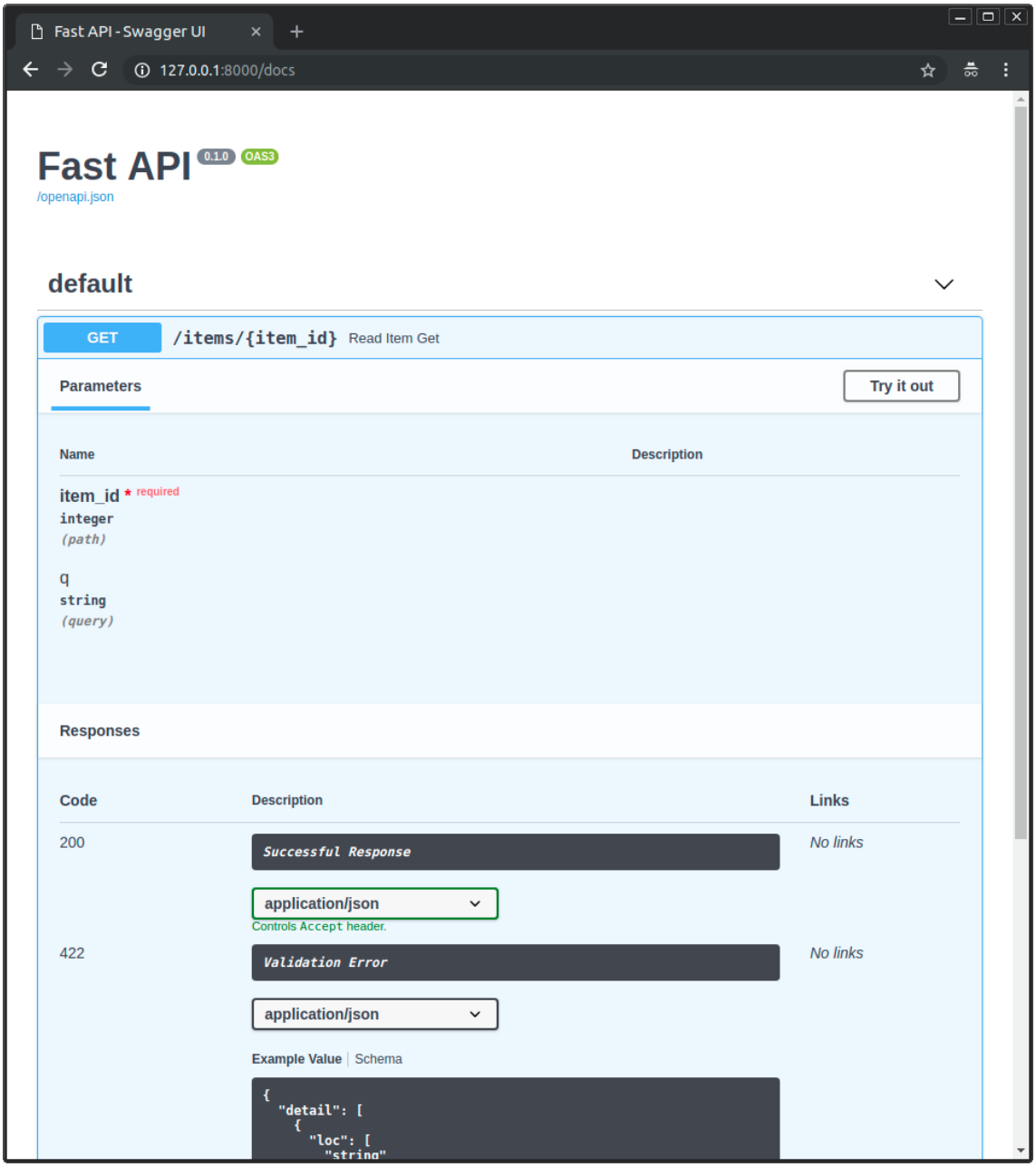
Verás la respuesta en JSON:

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

Documentación interactiva de la API

Ahora dirígete a http://127.0.0.1:8000/docs.

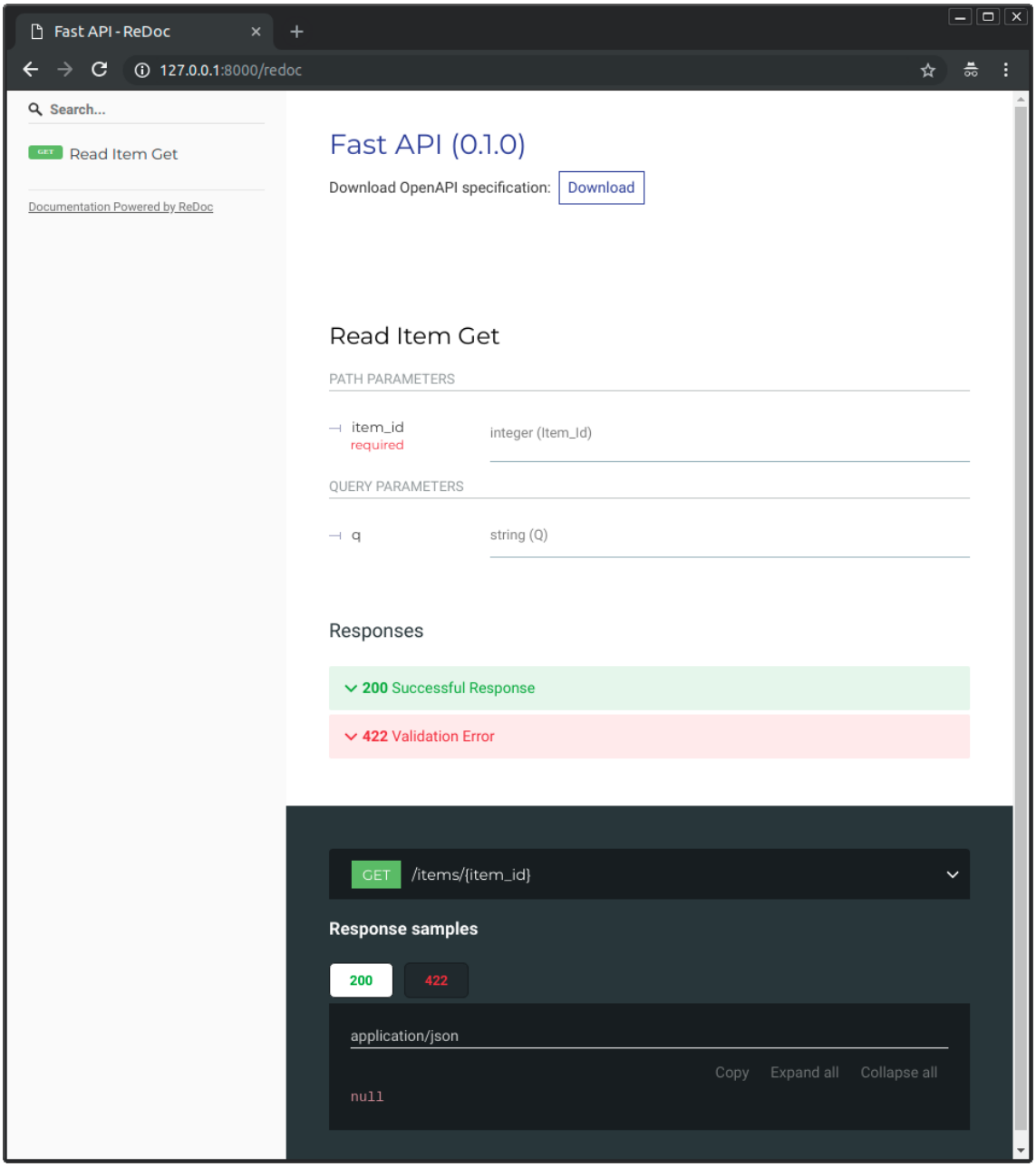
Ahí verás la documentación automática e interactiva de la API (proveída por Swagger UI):



Documentación alternativa de la API

Ahora, dirígete a http://127.0.0.1:8000/redoc.

Aquí verás la documentación automática alternativa (proveída por ReDoc):



OpenAPI

FastAPI genera un "schema" con toda tu API usando el estándar para definir APIs, **OpenAPI**.

"Schema"

Un "schema" es una definición o descripción de algo. No es el código que la implementa, sino solo una descripción abstracta.

"Schema" de la API

En este caso, OpenAPI es una especificación que dicta como se debe definir el schema de tu API.

La definición del schema incluye los paths de tu API, los parámetros que podría recibir, etc.

"Schema" de datos

El concepto "schema" también se puede referir a la forma de algunos datos, como un contenido en formato JSON.

En ese caso haría referencia a los atributos del JSON, los tipos de datos que tiene, etc.

OpenAPI y JSON Schema

OpenAPI define un schema de API para tu API. Ese schema incluye definiciones (o "schemas") de los datos enviados y recibidos por tu API usando **JSON Schema**, el estándar para los schemas de datos en JSON.

Revisa el `openapi.json`

Si sientes curiosidad por saber cómo se ve el schema de OpenAPI en bruto, FastAPI genera automáticamente un (schema) JSON con la descripción de todo tu API.

Lo puedes ver directamente en: http://127.0.0.1:8000/openapi.json.

Esto te mostrará un JSON que comienza con algo como:

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


...
```

¿Para qué se usa OpenAPI?

El schema de OpenAPI es lo que alimenta a los dos sistemas de documentación interactiva incluidos.

También hay docenas de alternativas, todas basadas en OpenAPI. Podrías añadir fácilmente cualquiera de esas alternativas a tu aplicación construida con **FastAPI**.

También podrías usarlo para generar código automáticamente, para los clientes que se comunican con tu API. Por ejemplo, frontend, móvil o aplicaciones de IoT.

Repaso, paso a paso

Paso 1: importa `FastAPI`

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

`FastAPI` es una clase de Python que provee toda la funcionalidad para tu API.

!!! note "Detalles Técnicos" `FastAPI` es una clase que hereda directamente de `Starlette`.

```
También puedes usar toda la funcionalidad de <a href="https://www.starlette.io/"
class="external-link" target="_blank">Starlette</a>.
```

Paso 2: crea un "instance" de `FastAPI`

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Aquí la variable `app` será un instance de la clase `FastAPI`.

Este será el punto de interacción principal para crear todo tu API.

Esta `app` es la misma a la que nos referimos cuando usamos el comando de `uvicorn`:

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Si creas un app como:

```
{!../../../docs_src/first_steps/tutorial002.py!}
```

y lo guardas en un archivo `main.py`, entonces ejecutarías `uvicorn` así:

```
$ uvicorn main:my_awesome_api --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Paso 3: crea un *operación de path*

Path

"Path" aquí se refiere a la última parte de una URL comenzando desde el primer `/`.

Entonces, en una URL como:

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

...el path sería:

```
/items/foo
```

!!! info "Información" Un "path" también se conoce habitualmente como "endpoint", "route" o "ruta".

Cuando construyes una API, el "path" es la manera principal de separar los "intereses" y los "recursos".

Operación

"Operación" aquí se refiere a uno de los "métodos" de HTTP.

Uno como:

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

...y los más exóticos:

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

En el protocolo de HTTP, te puedes comunicar con cada path usando uno (o más) de estos "métodos".

Cuando construyes APIs, normalmente usas uno de estos métodos específicos de HTTP para realizar una acción específica.

Normalmente usas:

- `POST` : para crear datos.
- `GET` : para leer datos.
- `PUT` : para actualizar datos.
- `DELETE` : para borrar datos.

Así que en OpenAPI, cada uno de estos métodos de HTTP es referido como una "operación".

Nosotros también los llamaremos "**operación**".

Define un *decorador de operaciones de path*

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

El `@app.get("/")` le dice a **FastAPI** que la función que tiene justo debajo está a cargo de manejar los requests que van a:

- el path `/`
- usando una operación `get`

!!! info "Información sobre `@decorator` " Esa sintaxis `@algo` se llama un "decorador" en Python.

```
Lo pones encima de una función. Es como un lindo sombrero decorado (creo que de ahí
salió el concepto).

Un "decorador" toma la función que tiene debajo y hace algo con ella.

En nuestro caso, este decorador le dice a **FastAPI** que la función que está debajo
corresponde al **path** `/` con una **operación** `get`.
```

```
Es el "**decorador de operaciones de path**".
```

También puedes usar las otras operaciones:

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

y las más exóticas:

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

!!! tip "Consejo" Tienes la libertad de usar cada operación (método de HTTP) como quieras.

```
**FastAPI** no impone ningún significado específico.

La información que está presentada aquí es una guía, no un requerimiento.

Por ejemplo, cuando usas GraphQL normalmente realizas todas las acciones usando
únicamente operaciones `POST`.
```

Paso 4: define la función de la operación de path

Esta es nuestra "**función de la operación de path**":

- **path**: es `/`.
- **operación**: es `get`.
- **función**: es la función debajo del "decorador" (debajo de `@app.get("/")`).

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Esto es una función de Python.

Esta función será llamada por **FastAPI** cada vez que reciba un request en la URL "`/`" usando una operación `GET`.

En este caso es una función `async`.

También podrías definirla como una función normal, en vez de `async def`:

```
{!../../../docs_src/first_steps/tutorial003.py!}
```

!!! note "Nota" Si no sabes la diferencia, revisa el Async: "*¿Tienes prisa?*"{.internal-link target=_blank}.

Paso 5: devuelve el contenido

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Puedes devolver `dict`, `list`, valores singulares como un `str`, `int`, etc.

También puedes devolver modelos de Pydantic (ya verás más sobre esto más adelante).

Hay muchos objetos y modelos que pueden ser convertidos automáticamente a JSON (incluyendo ORMs, etc.). Intenta usar tus favoritos, es muy probable que ya tengan soporte.

Repaso

- Importa `FastAPI`.
- Crea un instance de `app`.
- Escribe un **decorador de operación de path** (como `@app.get("/")`).
- Escribe una **función de la operación de path** (como `def root(): ...` arriba).
- Corre el servidor de desarrollo (como `uvicorn main:app --reload`).