



FastAPI framework, alto desempeño, fácil de aprender, rápido de programar, listo para producción



Documentación: <https://fastapi.tiangolo.com>

Código Fuente: <https://github.com/tiangolo/fastapi>



FastAPI es un web framework moderno y rápido (de alto rendimiento) para construir APIs con Python 3.6+ basado en las anotaciones de tipos estándar de Python.

Sus características principales son:

- **Rapidez:** Alto rendimiento, a la par con **NodeJS** y **Go** (gracias a Starlette y Pydantic). [Uno de los frameworks de Python más rápidos](#).
- **Rápido de programar:** Incrementa la velocidad de desarrollo entre 200% y 300%. *
- **Menos errores:** Reduce los errores humanos (de programador) aproximadamente un 40%. *
- **Intuitivo:** Gran soporte en los editores con auto completado en todas partes. Gasta menos tiempo debugging.
- **Fácil:** Está diseñado para ser fácil de usar y aprender. Gastando menos tiempo leyendo documentación.
- **Corto:** Minimiza la duplicación de código. Múltiples funcionalidades con cada declaración de parámetros. Menos errores.
- **Robusto:** Crea código listo para producción con documentación automática interactiva.
- **Basado en estándares:** Basado y totalmente compatible con los estándares abiertos para APIs: [OpenAPI](#) (conocido previamente como Swagger) y [JSON Schema](#).

* Esta estimación está basada en pruebas con un equipo de desarrollo interno contruyendo aplicaciones listas para producción.

Sponsors

{% if sponsors %} {% for sponsor in sponsors.gold -%}  {% endfor -%} {% for sponsor in sponsors.silver -%}  {% endfor %} {% endif %}

[Otros sponsors](#)

Opiniones

"[...] 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 [\(ref\)](#) - [\(ref\)](#)

Typer, el FastAPI de las CLIs



Si estás construyendo un app de CLI para ser usada en la terminal en vez de una API web, fíjate en [Typer](#).

Typer es el hermano menor de FastAPI. La intención es que sea el **FastAPI de las CLIs**.  

Requisitos

Python 3.6+

FastAPI está sobre los hombros de gigantes:

- [Starlette](#) para las partes web.

- [Pydantic](#) para las partes de datos.

Instalación

```
$ pip install fastapi

---> 100%
```

También vas a necesitar un servidor ASGI para producción cómo [Uvicorn](#) o [Hypercorn](#).

```
$ pip install uvicorn[standard]

---> 100%
```

Ejemplo

Créalo

- Crea un archivo `main.py` con:

```
from fastapi import FastAPI
from typing import Optional

app = FastAPI()

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

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

- O usa `async def...`

Córrelo

Corre el servidor con:

```
$ 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.
```

- Sobre el comando `uvicorn main:app --reload...`

Revísalo

Abre tu navegador en <http://127.0.0.1:8000/items/5?q=somequery>.

Verás la respuesta de JSON cómo:

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

Ya creaste una API que:

- Recibe HTTP requests en los *paths* `/` y `/items/{item_id}`.
- Ambos *paths* toman *operaciones* `GET` (también conocido como HTTP *methods*).
- El *path* `/items/{item_id}` tiene un *path parameter* `item_id` que debería ser un `int`.
- El *path* `/items/{item_id}` tiene un `str` *query parameter* `q` opcional.

Documentación interactiva de APIs

Ahora ve a <http://127.0.0.1:8000/docs>.

Verás la documentación automática e interactiva de la API (proveída por [Swagger UI](#)):

Fast API - Swagger UI

127.0.0.1:8000/docs

Fast API 0.1.0 OAS3

/openapi.json

default

GET /items/{item_id} Read Item Get

Try it out

Parameters

Name	Description
item_id <small>* required</small>	
integer	
(path)	
q	
string	
(query)	

Responses

Code	Description	Links
200	Successful Response	No links
	application/json	
	Controls Accept header.	
422	Validation Error	No links
	application/json	

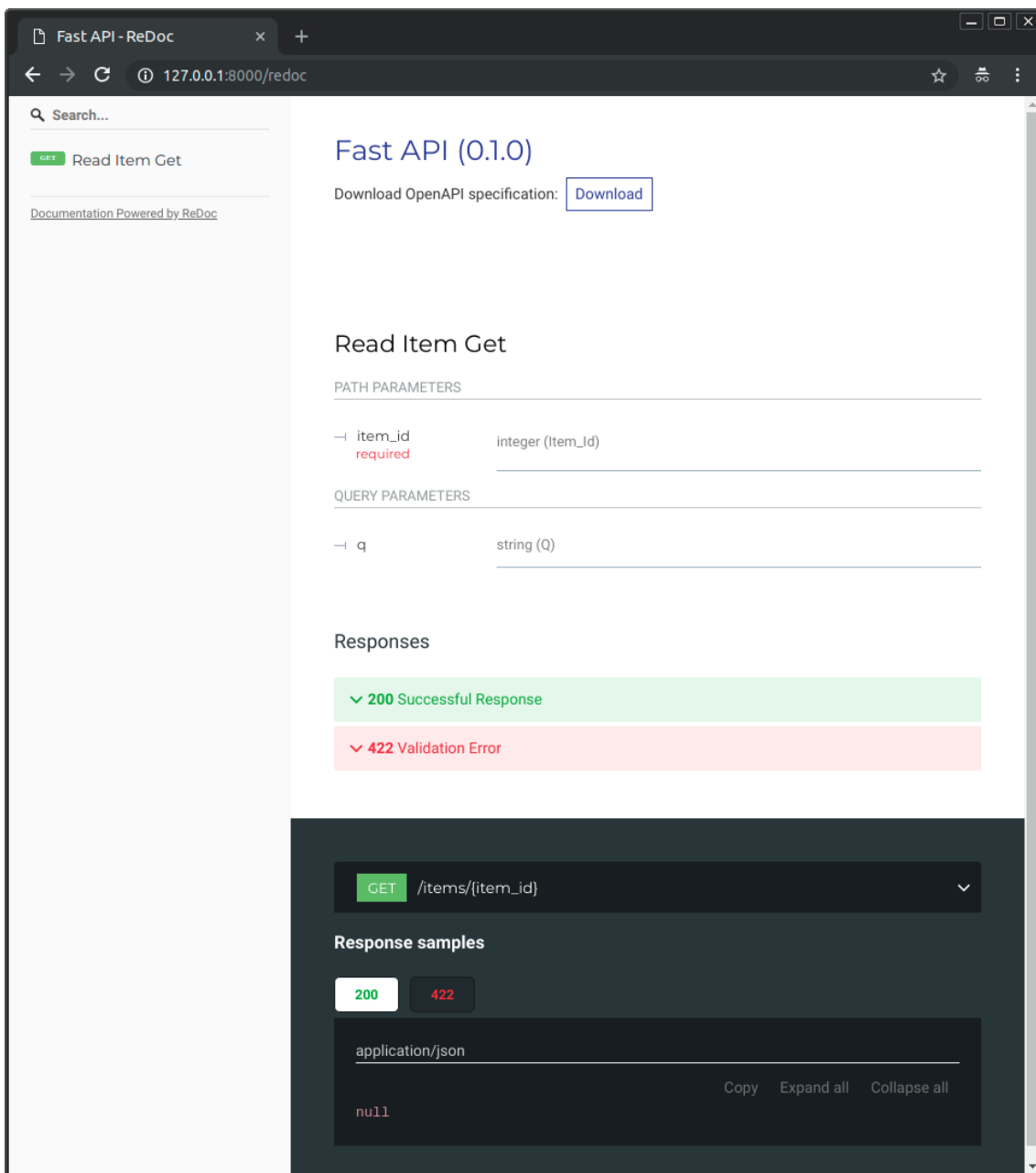
Example Value | Schema

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

Documentación alternativa de la API

Ahora, ve a <http://127.0.0.1:8000/redoc>.

Ahora verás la documentación automática alternativa (proveída por [ReDoc](#)):



Mejora al ejemplo

Ahora modifica el archivo `main.py` para recibir un body del `PUT` request.

Declara el body usando las declaraciones de tipo estándares de Python gracias a Pydantic.

```
from fastapi import FastAPI
from pydantic import BaseModel
from typing import Optional

app = FastAPI()
```

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

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

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Optional[str] = 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}
```

El servidor debería recargar automáticamente (porque añadiste `--reload` al comando `uvicorn` que está más arriba).

Mejora a la documentación interactiva de APIs

Ahora ve a <http://127.0.0.1:8000/docs>.

- La documentación interactiva de la API se actualizará automáticamente, incluyendo el nuevo body:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Fast API 0.1.0 OAS3

/openapi.json

default

GET / Read Root Get

GET /items/{item_id} Read Item Get

PUT /items/{item_id} Save Item Put

Parameters Try it out

Name Description

item_id * required
integer
(path)

Request body required application/json

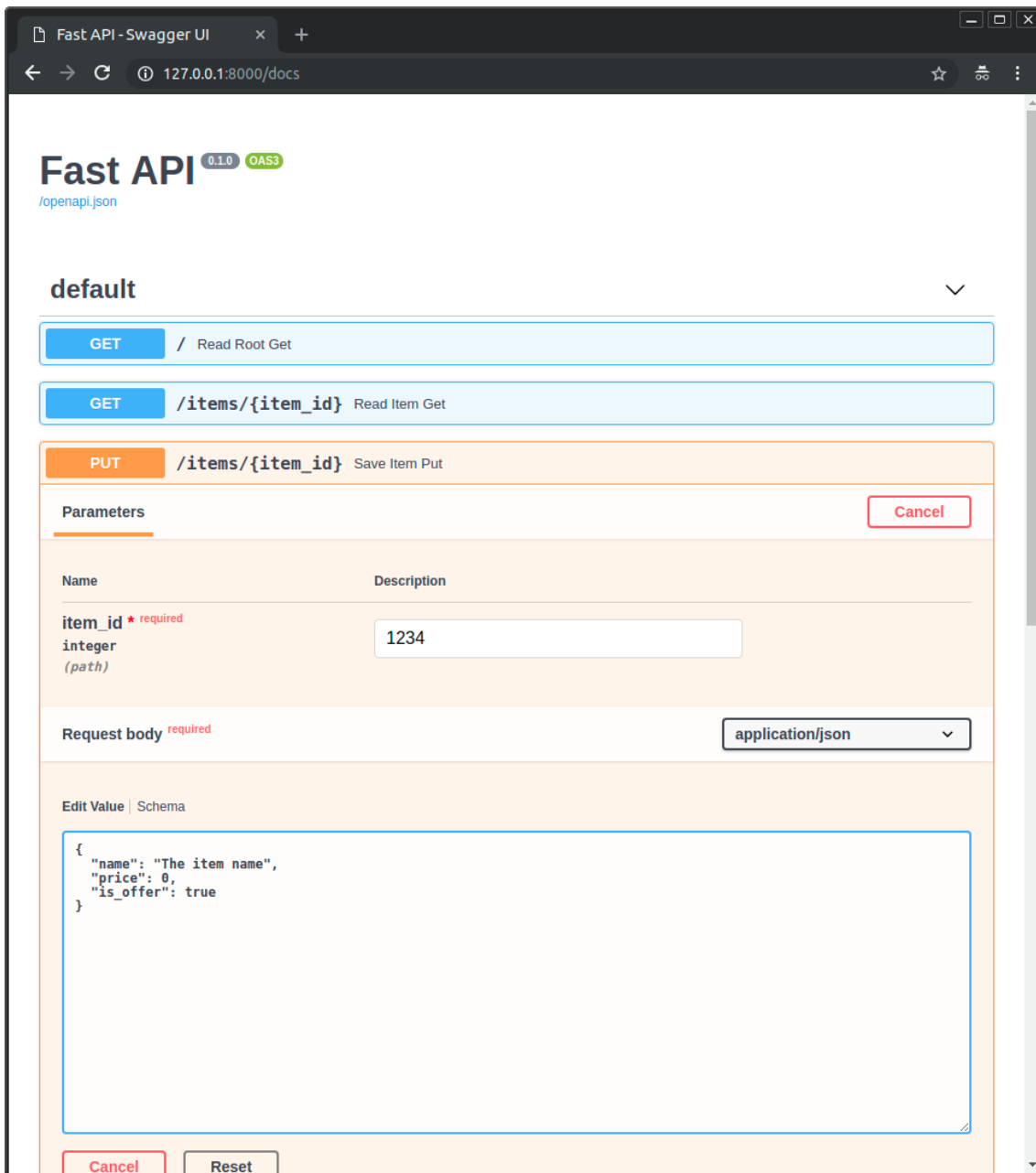
Example Value | Schema

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

Responses

Code	Description	Links
200	Successful Response	No links

- Haz clic en el botón de "Try it out" que te permite llenar los parámetros e interactuar directamente con la API:



- Luego haz clic en el botón de "Execute". La interfaz de usuario se comunicará con tu API, enviará los parámetros y recibirá los resultados para mostrarlos en pantalla:

Fast API - Swagger UI x +

127.0.0.1:8000/docs

Edit Value | Schema

```
{
  "name": "The item name",
  "price": 0,
  "is_offer": true
}
```

Cancel Reset

Execute Clear

Responses

Curl

```
curl -X PUT "http://127.0.0.1:8000/items/1234" -H "accept: application/json" -H "Content-Type: application/json" -d "{\"name\": \"The item name\", \"price\": 0, \"is_offer\": true}"
```

Request URL

```
http://127.0.0.1:8000/items/1234
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "item_name": "The item name", "item_id": 1234 }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 44 content-type: application/json</pre>

Mejora a la documentación alternativa de la API

Ahora, ve a <http://127.0.0.1:8000/redoc>.

- La documentación alternativa también reflejará el nuevo parámetro de query y el body:

Fast API - ReDoc

127.0.0.1:8000/redoc#operation/save_item_items__item_id__put

Search...

GET Read Root Get

GET Read Item Get

PUT Save Item Put

Documentation Powered by ReDoc

Save Item Put

PATH PARAMETERS

item_id	integer (Item_Id)
required	

REQUEST BODY SCHEMA: application/json

name	string (Name)
required	
price	number (Price)
required	
is_offer	boolean (Is_Offer)

Responses

- ✓ 200 Successful Response
- ✓ 422 Validation Error

PUT /items/{item_id}

Request samples

Payload

application/json

Copy Expand all Collapse all

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

Resumen

En resumen, declaras los tipos de parámetros, body, etc. **una vez** como parámetros de la función.

Lo haces con tipos modernos estándar de Python.

No tienes que aprender una sintaxis nueva, los métodos o clases de una library específica, etc.

Solo **Python 3.6+** estándar.

Por ejemplo, para un `int`:

```
item_id: int
```

o para un modelo más complejo de `Item` :

```
item: Item
```

...y con esa única declaración obtienes:

- Soporte del editor incluyendo:
 - Auto completado.
 - Anotaciones de tipos.
- Validación de datos:
 - Errores automáticos y claros cuándo los datos son inválidos.
 - Validación, incluso para objetos JSON profundamente anidados.
- Conversión de datos de input: viniendo de la red a datos y tipos de Python. Leyendo desde:
 - JSON.
 - Path parameters.
 - Query parameters.
 - Cookies.
 - Headers.
 - Formularios.
 - Archivos.
- Conversión de datos de output: convirtiendo de datos y tipos de Python a datos para la red (como JSON):
 - Convertir tipos de Python (`str` , `int` , `float` , `bool` , `list` , etc).
 - Objetos `datetime` .
 - Objetos `UUID` .
 - Modelos de bases de datos.
 - ...y muchos más.
- Documentación automática e interactiva incluyendo 2 interfaces de usuario alternativas:
 - Swagger UI.
 - ReDoc.

Volviendo al ejemplo de código anterior, **FastAPI** va a:

- Validar que existe un `item_id` en el path para requests usando `GET` y `PUT` .
- Validar que el `item_id` es del tipo `int` para requests de tipo `GET` y `PUT` .
 - Si no lo es, el cliente verá un mensaje de error útil y claro.
- Revisar si existe un query parameter opcional llamado `q` (cómo en `http://127.0.0.1:8000/items/foo?q=somequery`) para requests de tipo `GET` .
 - Como el parámetro `q` fue declarado con `= None` es opcional.
 - Sin el `None` sería obligatorio (cómo lo es el body en el caso con `PUT`).
- Para requests de tipo `PUT` a `/items/{item_id}` leer el body como JSON:
 - Revisar si tiene un atributo requerido `name` que debe ser un `str` .
 - Revisar si tiene un atributo requerido `price` que debe ser un `float` .
 - Revisar si tiene un atributo opcional `is_offer` , que debe ser un `bool` si está presente.
 - Todo esto funcionaría para objetos JSON profundamente anidados.
- Convertir de y a JSON automáticamente.
- Documentar todo con OpenAPI que puede ser usado por:
 - Sistemas de documentación interactiva.

- Sistemas de generación automática de código de cliente para muchos lenguajes.
- Proveer directamente 2 interfaces de documentación web interactivas.

Hasta ahora, escasamente vimos lo básico pero ya tienes una idea de cómo funciona.

Intenta cambiando la línea a:

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

...de:

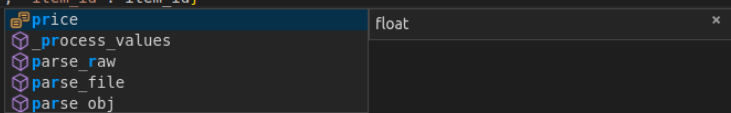
```
... "item_name": item.name ...
```

...a:

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

... y mira como el editor va a auto-completar los atributos y sabrá sus tipos:

```
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3
4  app = FastAPI()
5
6
7  class Item(BaseModel):
8      name: str
9      price: float
10     is_offer: bool = None
11
12
13 @app.get("/")
14 def read_root():
15     return {"Hello": "World"}
16
17
18 @app.get("/items/{item_id}")
19 def read_item(item_id: int, q: str = None):
20     return {"item_id": item_id, "q": q}
21
22
23 @app.put("/items/{item_id}")
24 def save_item(item_id: int, item: Item):
25     return {"item_name": item.pr, "item_id": item_id}
26
```



Para un ejemplo más completo que incluye más características ve el [Tutorial - Guía de Usuario](#).

Spoiler alert: el Tutorial - Guía de Usuario incluye:

- Declaración de **parámetros** en otros lugares diferentes como los: **headers**, **cookies**, **formularios** y **archivos**.
- Cómo agregar **requisitos de validación** como `maximum_length` o `regex`.
- Un sistema de **Dependency Injection** poderoso y fácil de usar.
- Seguridad y autenticación incluyendo soporte para **OAuth2** con **JWT tokens** y **HTTP Basic** auth.

- Técnicas más avanzadas, pero igual de fáciles, para declarar **modelos de JSON profundamente anidados** (gracias a Pydantic).
- Muchas características extra (gracias a Starlette) como:
 - **WebSockets**
 - **GraphQL**
 - pruebas extremadamente fáciles con `requests` y `pytest`
 - **CORS**
 - **Cookie Sessions**
 - ...y mucho más.

Rendimiento

Benchmarks independientes de TechEmpower muestran que aplicaciones de **FastAPI** corriendo con Uvicorn cómo [uno de los frameworks de Python más rápidos](#), únicamente debajo de Starlette y Uvicorn (usados internamente por FastAPI). (*)

Para entender más al respecto revisa la sección [Benchmarks](#).

Dependencias Opcionales

Usadas por Pydantic:

- [ujson](#) - para "parsing" de JSON más rápido.
- [email_validator](#) - para validación de emails.

Usados por Starlette:

- [requests](#) - Requerido si quieres usar el `TestClient`.
- [aiofiles](#) - Requerido si quieres usar `FileResponse` o `StaticFiles`.
- [jinja2](#) - Requerido si quieres usar la configuración por defecto de templates.
- [python-multipart](#) - Requerido si quieres dar soporte a "parsing" de formularios, con `request.form()`.
- [itsdangerous](#) - Requerido para dar soporte a `SessionMiddleware`.
- [pyyaml](#) - Requerido para dar soporte al `SchemaGenerator` de Starlette (probablemente no lo necesites con FastAPI).
- [graphene](#) - Requerido para dar soporte a `GraphQLApp`.
- [ujson](#) - Requerido si quieres usar `UJSONResponse`.

Usado por FastAPI / Starlette:

- [uvicorn](#) - para el servidor que carga y sirve tu aplicación.
- [orjson](#) - Requerido si quieres usar `ORJSONResponse`.

Puedes instalarlos con `pip install fastapi[all]`.

Licencia

Este proyecto está licenciado bajo los términos de la licencia del MIT.