

공개SW솔루션 설치 & 활용 가이드

시스템SW > SW공학도구



FastAPI

제대로 배워보자

How to Use Open Source Software

Open Source Software Installation & Application Guide



오픈소스 소프트웨어 통합지원센터
Open Source Software Support Center



CONTENTS

1. 개요
2. 기능요약
3. 설치 및 실행
4. 기능소개
5. 활용예제
6. FAQ
7. 용어정리

1. 개요



소개	<ul style="list-style-type: none">• FastAPI는 현대적이고, 빠르며(고성능), 파이썬 표준 타입 힌트에 기초한 Python3.6+의 API를 빌드하기 위한 웹 프레임워크• NodeJS 및 Go와 대등할 정도로 매우 높은 성능을 가짐• 사용 가능한 가장 빠른 파이썬 프레임워크 중 하나임		
주요기능	<ul style="list-style-type: none">• 웹 서버 WSGI (Web Server Gateway Interface) 지원• 비동기 ASGI (Asynchronous Server Gateway Interface)을 지원		
라이선스형태	• MIT	사전설치 솔루션	• Python 3.6 +
실행 하드웨어	• Python 3.6 버전이 구동 될 수 있는 환경 (1 GHz 프로세서, 4GB 메모리 이상 권장)	버전	• 0.88.0(2022년 11월)
특징	<ul style="list-style-type: none">• 빠른 코드 작성: 약 200%에서 300%까지 기능 개발 속도 증가• 적은 버그: 사람(개발자)에 의한 에러 약 40% 감소• 직관적: 훌륭한 편집기 지원. 모든 곳에서 자동완성. 적은 디버깅 시간		
취약점	<ul style="list-style-type: none">• 표준화되어 있지 않기때문에 Django 등의 프레임워크 전환이 어려울 수 있음• 개발 확장과 배포를 위해서 라이브러리 및 확장을 많이 검색해봐야 함• 대체 프레임워크들(Django, Flask)과 비교했을 때, 데이터베이스와의 연동이 다소 어려움		
개발자/커뮤니티	• https://gitter.im/fast_api/community		
공식 홈페이지	• https://fastapi.tiangolo.com		



2. 기능요약



- FastAPI 의 주요 기능

주요기능	기능설명
API 생성	다른 파이썬 프레임워크(Django, Flask)등에 비해 API를 생성하는 것이 매우 용이하며 기술명 자체에 API가 들어있는 만큼 API를 다루는데에 최적화된 기능을 제공
WebSocket	웹 소켓은 사용자의 브라우저와 서버 사이의 인터랙티브 통신 세션을 설정할 수 있게 하는 기술임 개발자는 웹 소켓 API를 통해 서버로 메시지를 보내고 서버의 응답을 위해 서버를 폴링하지 않고도 이벤트 중심 응답을 받는 것이 가능
CORS 지원	CORS (Cross-Origin Resource Sharing)는 HTTP 헤더 기반 메커니즘으로, 브라우저가 리소스 로드를 허용해야 하는 자체 출처 이외의 모든 출처 (도메인, 스키마 또는 포트)를 서버가 표시할 수 있도록 함



3. 설치 및 실행



- 설치 실습 환경

- 운영체제: Ubuntu 22.04 LTS



- 파이썬 버전: 3.10.6



- 코드 편집기: Visual Studio Code



3. 설치 및 실행



- 설치
 - 터미널 실행 후 설치 명령어 입력

```
$ pip install fastapi
```

- ASGI server 설치

```
$ pip install "uvicorn[standard]"
```



3. 설치 및 실행



- 실행
 - main.py를 만들어서 아래 코드를 입력

```
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}
```



3. 설치 및 실행



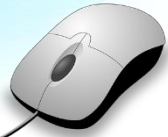
- 실행
 - \$ `uvicorn main:app --reload` 명령어로 실행

```
INFO: Will watch for changes in these directories: ['/home/hwan/Desktop/fast_api_test']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [172641] using WatchFiles
INFO: Started server process [172643]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

- 위와 같이 실행되면 <http://127.0.0.1:8000>으로 접속해서 확인



3. 설치 및 실행



- 실행

- \$ `uvicorn main:app --reload` 명령어로 실행



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

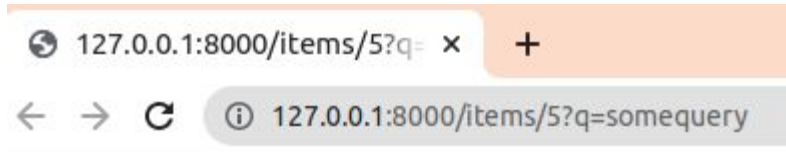
- 위와 같이 실행되면 <http://127.0.0.1:8000>으로 접속해서 확인





- 실행

- <http://127.0.0.1:8000/items/5?q=somequery> 접속 확인



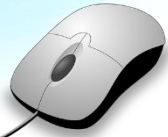
```
{"item_id":5,"q":"somequery"}
```

- 위 결과물은 main.py에 생성된 아래 코드를 통해 생성된 API response

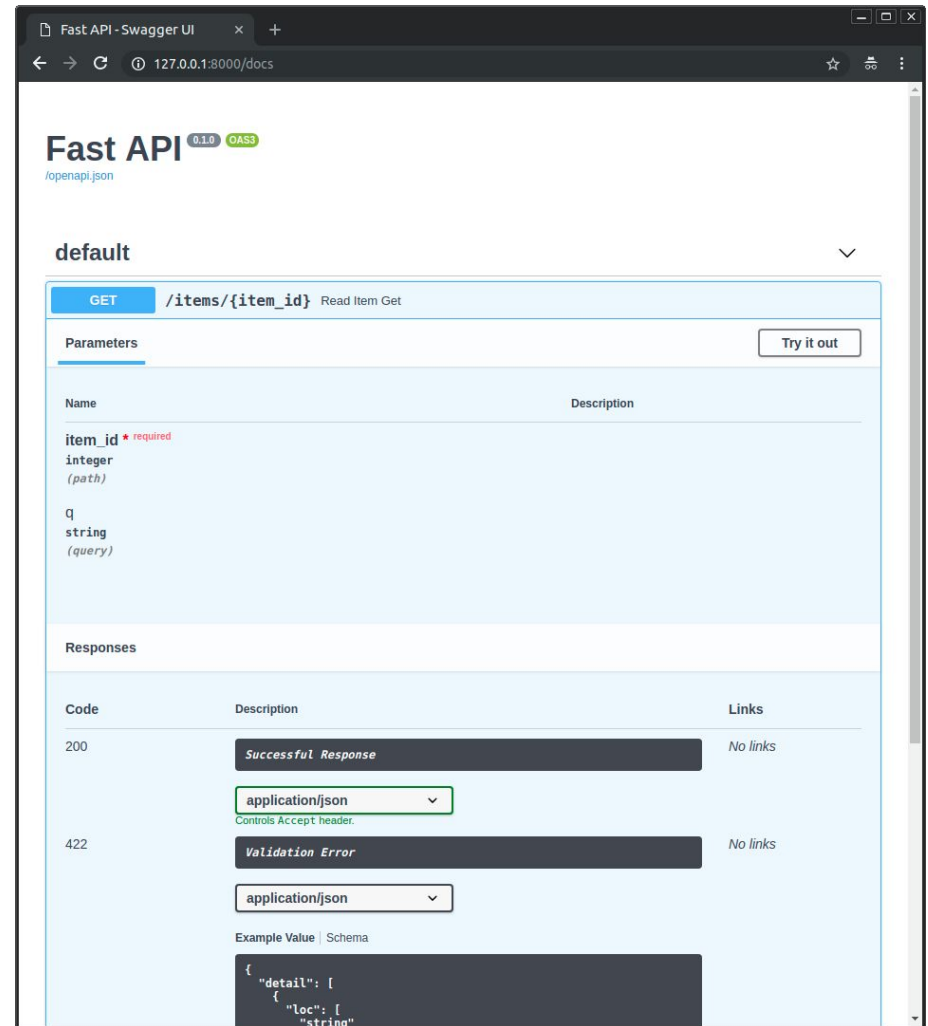
```
@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```



3. 설치 및 실행



- 실행
 - 자동 생성된 API 문서 확인
<http://127.0.0.1:8000/docs>



The screenshot shows the Swagger UI for a FastAPI application. The browser address bar shows the URL `127.0.0.1:8000/docs`. The page title is "Fast API 0.1.0 OAS3". The API endpoint is `GET /items/{item_id}` with the description "Read Item Get".

Parameters

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

Responses

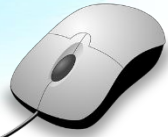
Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

Example Value | Schema

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



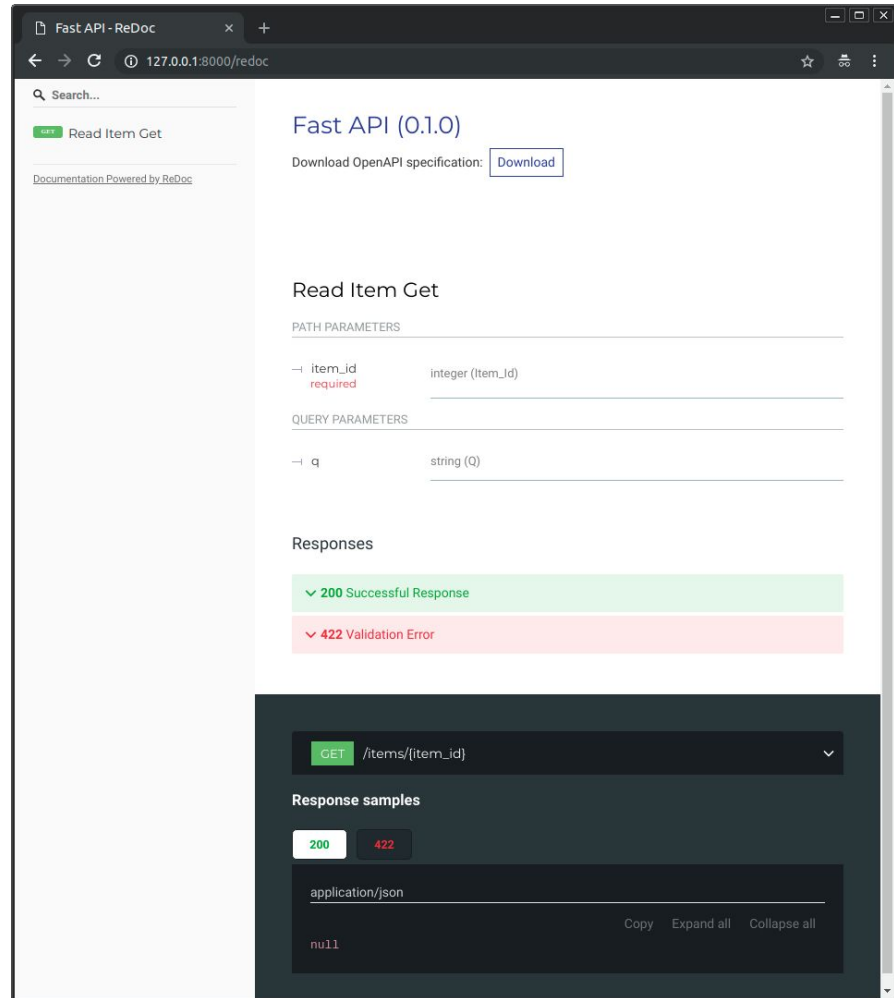
3. 설치 및 실행



- 실행

- 자동 생성된 API 문서 확인

- <http://127.0.0.1:8000/redoc>



4. 기능소개



4.1 WebSocket

- WebSocket은 client와 server 사이에 communication channel을 열어서 실시간 소통 가능
- HTTP 통신은 요청을 보내면 응답을 받는 단방향 통신만 가능했지만 WebSocket은 TCP 기반으로 양방향 통신 가능
- FastAPI에서는 이 웹소켓을 간단히 생성 가능



4. 기능소개



4.1 WebSocket

- main.py에 아래 코드를 입력 (1/4)

```
from fastapi import FastAPI, WebSocket
from fastapi.responses import HTMLResponse

app = FastAPI()
```



4. 기능소개



4.1 WebSocket

- main.py에 아래 코드를 입력 (2/4)

```
html = """
<body>
  <h1>WebSocket Chat</h1>
  <form action="" onsubmit="sendMessage(event)">
    <input type="text" id="messageText"
autocomplete="off"/>
    <button>Send</button>
  </form>
  <ul id='messages'>
  </ul>

```





4.1 WebSocket

- main.py에 아래 코드를 입력 (3/4)

```
<script>
  var ws = new WebSocket("ws://localhost:8000/ws");
  ws.onmessage = function(event) {
    var messages = document.getElementById('messages')
    var message = document.createElement('li')
    var content = document.createTextNode(event.data)
    message.appendChild(content)
    messages.appendChild(message)
  };
  function sendMessage(event) {
    var input = document.getElementById("messageText")
    ws.send(input.value)
    input.value = ''
    event.preventDefault()
  }
</script>
</body>
"""
```





4.1 WebSocket

- main.py에 아래 코드를 입력 (4/4)

```
@app.get("/")
async def get():
    return HTMLResponse(html)

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
        data = await websocket.receive_text()
        await websocket.send_text(f"Message text was: {data}")
```

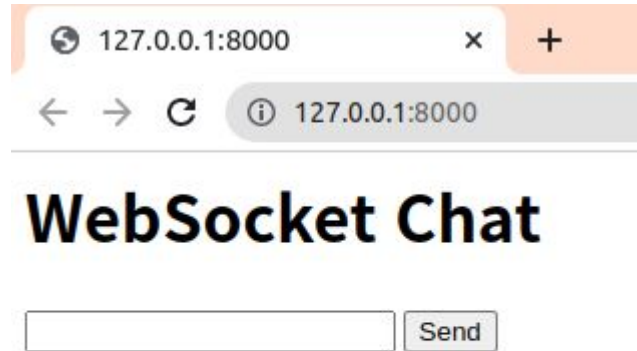


4. 기능소개



4.1 WebSocket

- 실행: `$ uvicorn main:app --reload`
 - <http://127.0.0.1:8000> 접속해서 확인



4. 기능소개



4.1 WebSocket

- 실행: `$ uvicorn main:app --reload`
 - <http://127.0.0.1:8000> 접속해서 확인



WebSocket Chat

WebSocket Chat

- Message text was: Hi OSS



4. 기능소개



4.2 CORS 지원

- FastAPI의 CORSMiddleware를 사용하여 CORS를 설정 가능
- 자격 증명 허용 여부 지정(인증, 헤더, 쿠키 등) 가능



4.2 CORS 지원

- 코드 예제

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = ["http://localhost.tiangolo.com",
           "https://localhost.tiangolo.com",
           "http://localhost", "http://localhost:8080",]

app.add_middleware(CORSMiddleware, allow_origins=origins,
                  allow_credentials=True, allow_methods=["*"], allow_headers=["*"],)

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





- 실행
 - main.py에 아래 코드 입력

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

app = FastAPI()

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

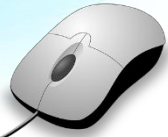
@app.get("/")
def read_root():
    return {"Hello": "World"}

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

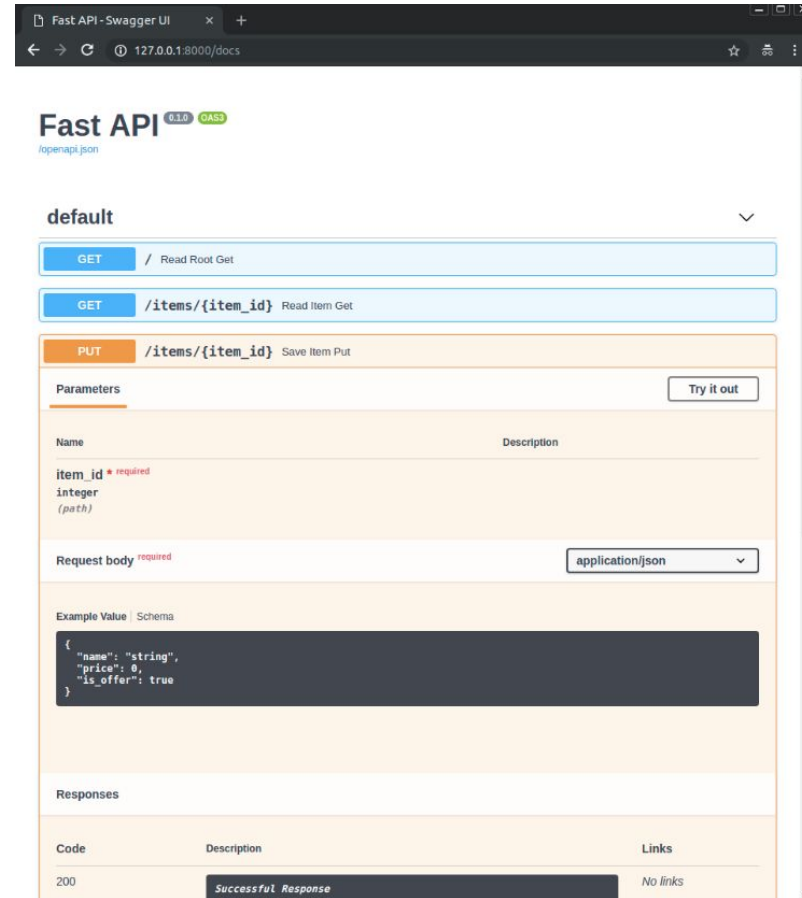
@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}
```



5. 활용예제



- 실행
 - API 설명서 자동 업데이트
<http://127.0.0.1:8000/docs>



5. 활용예제

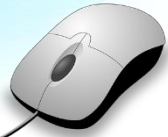


- 실행
 - Try it out 버튼 클릭

The screenshot shows a REST client interface for a PUT request. The top bar displays the HTTP method 'PUT' in an orange box, followed by the endpoint path '/items/{item_id}' and the description 'Update Item'. On the right side of this bar is an upward-pointing chevron icon. Below the top bar, the word 'Parameters' is visible on the left, and a rectangular button labeled 'Try it out' is positioned on the right.



5. 활용예제



- 실행
 - Try it out 버튼 클릭 후 “10” 입력 후 request 보내기 (Execute 클릭)

PUT /items/{item_id} Update Item

Parameters Cancel

Name	Description
item_id * required	
integer (path)	10

Request body required application/json

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

Execute Clear



5. 활용예제



- 실행
 - Response 확인

Responses

Curl

```
curl -X 'PUT' \  
'http://127.0.0.1:8000/items/10' \  
-H 'accept: application/json' \  
-H 'Content-Type: application/json' \  
-d '{  
  "name": "string",  
  "price": 0,  
  "is_offer": true  
}'
```

Request URL

```
http://127.0.0.1:8000/items/10
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "item_name": "string", "item_id": 10 }</pre> <p>Response headers</p> <pre>content-length: 35 content-type: application/json date: Tue,20 Dec 2022 08:23:57 GMT server: uvicorn</pre>





- 실행
 - 스크립트 작성해서 python code로 실행해보기
 - requests 설치 (HTTP 통신할 수 있는 python 내장 library)

```
pip install requests
```

- test.py 파일을 만들어서 아래와 같이 스크립트를 작성해서 실행

```
import requests

res = requests.get("http://127.0.0.1:8000/")
res2 = requests.get("http://127.0.0.1:8000/items/10?q=10")

res_json = res.json()
res_json2 = res2.json()

print(res_json)
print(res_json2)
```



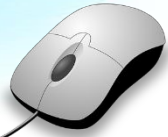
5. 활용예제



- 실행
 - 스크립트 작성해서 python code로 실행해보기
 - 결과값

```
{'Hello': 'World'}  
{'item_id': 10, 'q': '10'}
```





Q django, flask 가 있는데 왜 FastAPI를 써야 하나요?

A ASGI인 Uvicorn을 사용함으로 성능이 우수합니다. 구체적인 벤치마킹 점수를 보면 Flask, Django가 1,000점대의 점수를 갖는 것에 비해, FastAPI는 14,000점대의 점수가 나올 정도로 성능이 우수합니다.

Q 어떻게 성능차이가 그렇게 날 수가 있나요?

A FastAPI가 비동기 프로그래밍에 기반한 동시성 제어 모델을 사용하기 때문입니다. Flask 환경에서는 thread가 작업을 기다리고 있는것에 반해서 FastAPI 환경에서는 thread가 대기 시간동안 다른 일을 처리할 수 있어서 Flask에 비해서도 자원을 효율적으로 사용할 수 있습니다.





용어	설명
<pre>\$ uvicorn main:app -reload</pre>	<p>uvicorn: 매우 가벼운 ASGI 서버 해당 명령어에서는 uvicorn에 해당하는 명령어의 시작을 의미</p> <p>main: main.py 파일을 지칭</p> <p>app: main.py 내부에 app=FastAPI()로 생성된 개체</p> <p>--reload : 코드가 변경된 후에 서버를 다시 시작(개발환경에서만 사용)</p>
API	API는 정의 및 프로토콜 집합을 사용하여 두 소프트웨어 구성 요소가 서로 통신할 수 있게 하는 메커니즘



Open Source Software Installation & Application Guide



이 저작물은크리에이티브커먼즈[저작자표시- 비영리- 동일조건변경허락 2 . 0 대한민국라이선스]에 따라
이용하실 수 있습니다.