Serving ML easily with

High performance, easy to learn,
fast to code, ready for production

FastAPI for ML people

I will assume you know:

- Machine Learning
- The basics of web / API development
- HTTP, JSON...

You have an ML model

```
import spacy

nlp_en = spacy.load("en_core_web_sm")
doc_en = nlp_en("Apple buys U.K. startup for $1 billion")

for ent in doc_en.ents:
    print(ent.text, ent.label_)

# Apple ORG
# U.K. GPE
# $1 billion MONEY
```

Make it a web API

- Show your colleagues
- Integrate with other internal apps
- Deploy to production

Basic **FastAPI** app

```python
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: str = None):
    return {"item_id": item_id, "q": q}
```

https://somedomain.com/items/5?q=some+query

@tiangolo

Basic **FastAPI** app - docs

Fast API 0.1.0 OAS3

/openapi.json

https://somedomain.com/docs

default

GET / Read Root

GET /items/{item_id} Read Item

Parameters

Try it out

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

@tiangolo

Basic **FastAPI** app with body

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Data(BaseModel):
    text: str
    lang: str


@app.post("/text/")
def extract_entities(data: Data):
    return {"message": data.text}
```

@tiangolo

Basic **FastAPI** app with body - docs

Fast API 0.1.0 OAS3

/openapi.json

default

POST /text/ Extract Entities

Parameters | Try it out

No parameters

Request body required | application/json

Example Value | Schema

```
{
  "text": "string",
  "lang": "string"
}
```

Responses

Code	Description	Links
200		No links

@tiangolo

Basic **FastAPI** app with body - docs

Request body required

application/json

```
{
  "text": "Apple buys U.K. startup for $1 billion",
  "lang": "en"
}
```

Execute | Clear

Responses

Curl

```
curl -X POST "http://localhost:8000/text/" -H "accept: application/json" -H "Content-Type: application/json" -d "
{\"text\":\"Apple buys U.K. startup for $1 billion\",\"lang\":\"en\"}"
```

Request URL

```
http://localhost:8000/text/
```

Server response

Code	Details
200	Response body `{ "message": "Apple buys U.K. startup for $1 billion" }` Download Response headers

@tiangolo

Basic **FastAPI** app with body and query

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Data(BaseModel):
    text: str


@app.post("/text/")
def extract_entities(data: Data, lang: str):
    return {"message": data.text, "lang": lang}
```

@tiangolo

Basic **FastAPI** app with body and query - docs

default

POST `/text/` Extract Entities

Parameters | Try it out

Name	Description
lang * required string *(query)*	lang

Request body required — application/json

Example Value | Schema

```
{
  "text": "string"
}
```

@tiangolo

FastAPI app with ML

```python
from fastapi import FastAPI
from pydantic import BaseModel
import spacy

nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


class Data(BaseModel):
    text: str


@app.post("/text/")
def extract_entities(data: Data, lang: str):
    doc_en = nlp_en(data.text)
    ents = []
    for ent in doc_en.ents:
        ents.append({"text": ent.text, "label_": ent.label_})
    return {"message": data.text, "lang": lang, "ents": ents}
```

@tiangolo

FastAPI app with ML - docs

Request URL

```
http://localhost:8000/text/?lang=en
```

Server response

Code	Details
200	Response body


```
{
  "message": "Apple buys U.K. startup for $1 billion",
  "lang": "en",
  "ents": [
    {
      "text": "Apple",
      "label_": "ORG"
    },
    {
      "text": "U.K.",
      "label_": "GPE"
    },
    {
      "text": "$1 billion",
      "label_": "MONEY"
    }
  ]
}
```


Download

@tiangolo

JSON array of objects in body

```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel
import spacy

nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


class Data(BaseModel):
    text: str


@app.post("/text/")
def extract_entities(data: List[Data], lang: str):
    response_body = []
    for item in data:
        doc_en = nlp_en(item.text)
        ents = []
        for ent in doc_en.ents:
            ents.append({"text": ent.text, "label_": ent.label_})
        response_body.append({"message": item.text, "ents": ents})
    return response_body
```

@tiangolo

JSON array of objects in body - docs

default

POST `/text/` Extract Entities

Parameters | Try it out

Name	Description
lang * required string *(query)*	lang

Request body required — application/json

Example Value | Schema

```
[
  {
    "text": "string"
  }
]
```

@tiangolo

JSON array of objects in body - invalid data

Request body required

application/json

```
[
  {
    "text": "Apple buys U.K. startup for $1 billion"
  },
  {
    "text": "PyCon attendees arrived to Minsk"
  },
  {
    "content": "Invalid data won't be accepted"
  },
  {
    "text": {"content": "Even in deeply nested data structures"}
  }
]
```

Execute | Clear

JSON array of objects in body - validation

Server response

Code	Details
422	Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "loc": [
        "body",
        "bodies",
        2,
        "text"
      ],
      "msg": "field required",
      "type": "value_error.missing"
    },
    {
      "loc": [
        "body",
        "bodies",
        3,
        "text"
      ],
      "msg": "str type expected",
      "type": "type_error.str"
    }
  ]
}
```


Download

@tiangolo

Type checks

```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Data(BaseModel):
    id: int
    text: str


@app.post("/text/")
def extract_entities(data: List[Data], lang: str):
    response_body = []
    for item in data:
        message = item.text + item.id
        response_body.append({"message": message})
    return response_body
```

@tiangolo

Type checks - errors

```python
from typing import List

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Data(BaseModel):
    id: int
    text: str


@app.post("/text/")
def extract_entities(data: Lis
    response_body = []
    for item in data:
        message = item.text + item.id
        response_body.append({"message": message})
    return response_body
```

```
item: Data
Unsupported operand types for + ("str" and "int") my
Peek Problem    No quick fixes available
```

@tiangolo

Upload a file

```
from fastapi import FastAPI, UploadFile, File
import spacy

nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


@app.post("/file/")
def extract_entities(file: UploadFile = File(...)):
    content_bytes = file.file.read()
    content_text = content_bytes.decode()
    doc_en = nlp_en(content_text)
    ents = []
    for ent in doc_en.ents:
        ents.append({"text": ent.text, "label_": ent.label_})
    return {"message": content_text, "ents": ents}
```

@tiangolo

Upload a file - docs

POST /file/ Extract Entities

Parameters

Cancel

No parameters

Request body required

multipart/form-data

file * required
string($binary)

Choose File testfile.txt

Execute

Clear

@tiangolo

FastAPI - WebSockets

- Real-time
- Chats (really?)
- Video
- Synchronized interactive animations
- Synchronized text typing (live coding)

FastAPI - WebSockets

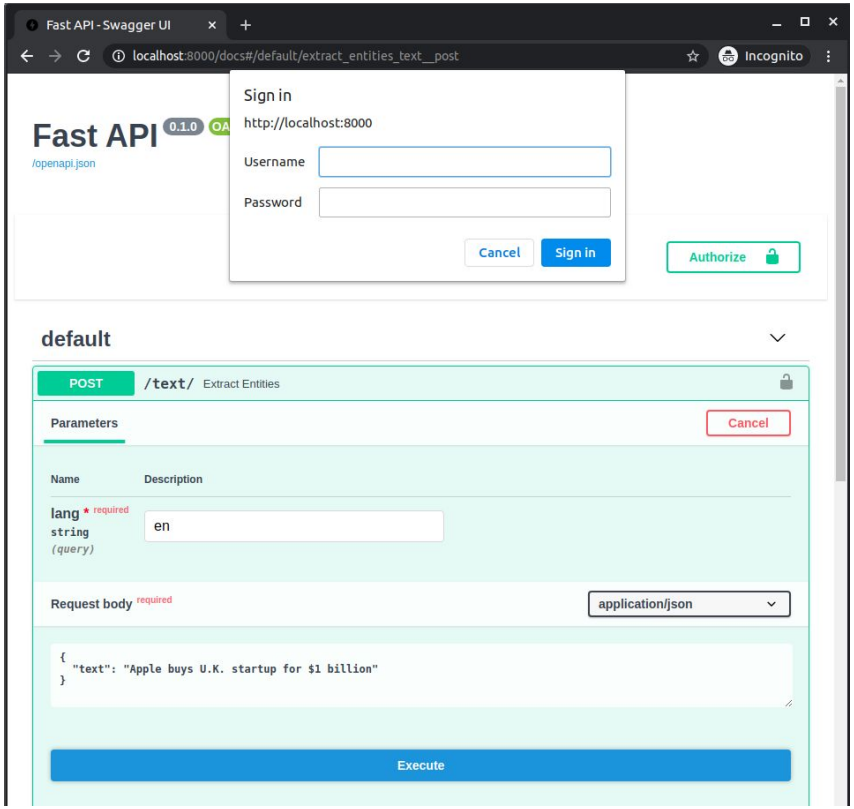
```python
from fastapi import FastAPI
from starlette.websockets import WebSocket
import spacy

nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
        data = await websocket.receive_text()
        doc_en = nlp_en(data)
        ents = []
        for ent in doc_en.ents:
            ents.append({"text": ent.text, "label_": ent.label_})
        await websocket.send_json(ents)
```

@tiangolo

FastAPI - HTTP Basic Auth



@tiangolo

FastAPI - HTTP Basic Auth

```python
import secrets

from fastapi import FastAPI, Depends, HTTPException
from fastapi.security import HTTPBasic, HTTPBasicCredentials
from pydantic import BaseModel
import spacy

security = HTTPBasic()


def get_current_user(credentials: HTTPBasicCredentials = Depends(security)):
    correct_username = secrets.compare_digest(credentials.username, "stanleyjobson")
    correct_password = secrets.compare_digest(credentials.password, "swordfish")
    if not (correct_username and correct_password):
        raise HTTPException(
            status_code=401,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Basic"},
        )
    return credentials.username
```

@tiangolo

FastAPI - HTTP Basic Auth, part 2

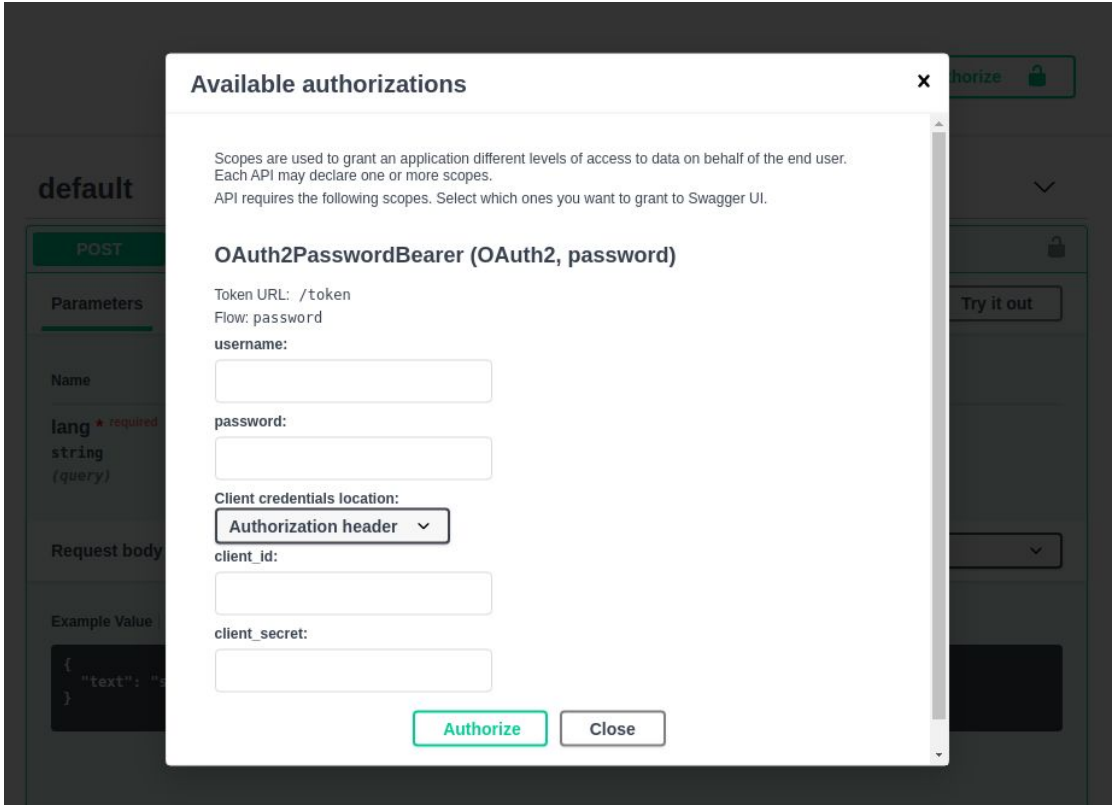
```python
nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


class Data(BaseModel):
    text: str


@app.post("/text/")
def extract_entities(data: Data, lang: str, username: str = Depends(get_current_user))
    doc_en = nlp_en(data.text)
    ents = []
    for ent in doc_en.ents:
        ents.append({"text": ent.text, "label_": ent.label_})
    return {"message": data.text, "lang": lang, "ents": ents, "username": username}
```

@tiangolo

FastAPI - OAuth2



@tiangolo

FastAPI - OAuth2

```python
import secrets

import spacy
from fastapi import Depends, FastAPI, HTTPException
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from pydantic import BaseModel

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/token")


async def get_current_user(token: str = Depends(oauth2_scheme)):
    if not token == "stanleyjobson":
        raise HTTPException(
            status_code=401,
            detail="Invalid authentication credentials",
            headers={"WWW-Authenticate": "Bearer"},
        )
    return token


class Data(BaseModel):
    text: str
```

@tiangolo

FastAPI - OAuth2, part 2

```python
nlp_en = spacy.load("en_core_web_sm")
app = FastAPI()


@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    correct_username = secrets.compare_digest(form_data.username, "stanleyjobson")
    correct_password = secrets.compare_digest(form_data.password, "swordfish")
    if not (correct_username and correct_password):
        raise HTTPException(status_code=400, detail="Incorrect email or password")
    return {"access_token": form_data.username, "token_type": "bearer"}


@app.post("/text/")
def extract_entities(data: Data, lang: str, username: str = Depends(get_current_user))
    doc_en = nlp_en(data.text)
    ents = []
    for ent in doc_en.ents:
        ents.append({"text": ent.text, "label_": ent.label_})
    return {"message": data.text, "lang": lang, "ents": ents}
```

@tiangolo