**Aсинхронный API сервис**  
  
from fastapi import FastAPI

from pydantic import BaseModel

app = FastAPI()

class LogEntry(BaseModel):

ip\_address: str

http\_method: str

uri: str

status\_code: int

@app.post("/log")

async def log\_entry(entry: LogEntry):

return {"message": "Log entry saved successfully"}

from fastapi import FastAPI

app = FastAPI()

@app.get("/logs/{log\_id}")

async def get\_log\_entry(log\_id: int):

return {"log\_id": log\_id, "message": "Log entry retrieved successfully"}

**Фоновый сервер?**

import time

def background\_task():

while True:

time.sleep(60) # Задержка в 60 секунд

import threading

thread = threading.Thread(target=background\_task)

thread.start()

**Dockerfile для клиента:**

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

**Dockerfile для фона:**

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "main.py"]

docker-compose.yml:

version: "3"

services:

client:

build:

context: ./client

ports:

- "8000:8000"

web:

build:

context: ./web

ports:

- "8001:8000"

background:

build:

context: ./background

**Спецификация эндпоинтов.**

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

from datetime import datetime

from uuid import UUID, uuid4

from ipaddress import IPv4Address

app = FastAPI()

class Log(BaseModel):

ip: IPv4Address

method: str

uri: str

status\_code: int

class LogEntry(BaseModel):

id: UUID

created: datetime

log: Log

database = []

@app.post("/api/data", status\_code=201)

async def save\_log\_entry(log: str):

try:

ip, method, uri, status\_code = log.split()

log\_entry = Log(ip=ip, method=method, uri=uri, status\_code=int(status\_code))

entry\_id = uuid4()

created = datetime.now()

log\_entry = LogEntry(id=entry\_id, created=created, log=log\_entry)

database.append(log\_entry)

return {"message": "Лог сохранен"}

except:

raise HTTPException(status\_code=418, detail="Что-то пошло не так")

@app.get("/api/data")

async def get\_log\_entries():

return database

**Попытка обновить код**

import requests

import random

import time

import logging

import os

from threading import Thread

logging.basicConfig(filename='generated\_logs.log', level=logging.INFO, format='%(asctime)s - %(message)s')

N = int(os.getenv('NUM\_THREADS', '1'))

M = int(os.getenv('DELAY\_MS', '1000'))

def generate\_log():

ip = '192.168.0.1' # Пример IP-адреса

http\_methods = ['GET', 'POST', 'PUT', 'DELETE']

uris = ['/api/data', '/api/user', '/api/product']

status\_codes = [200, 201, 400, 404, 500]

while True:

Генерация '{IP адрес} {HTTP method} {URI} {HTTP status code}'

log\_entry = f'{ip} {random.choice(http\_methods)} {random.choice(uris)} {random.choice(status\_codes)}'

**POST-запрос**

response = requests.post('http://web:8001/log', json={'log': log\_entry})

if response.status\_code == 201:

logging.info(log\_entry)

time.sleep(M / 1000)

**N потоки/процессов**

threads = []

for \_ in range(N):

thread = Thread(target=generate\_log)

thread.start()

threads.append(thread)

for thread in threads:

thread.join()