**✅ 권장 FastAPI 프로젝트 구조 (AiGov-Proxy 기준)**

css

복사편집

AiGov-Proxy/

├── main.py ← 앱 진입점 (uvicorn 실행용)

├── app/ ← 모든 로직과 모듈의 집합

│ ├── \_\_init\_\_.py

│ ├── api/ ← 라우터(API 엔드포인트)

│ │ └── proxy.py

│ ├── core/ ← 핵심 로직 (필터, 유효성검사 등)

│ │ ├── filter.py ← SensitiveDataFilter 클래스

│ │ └── validator.py ← InputValidator 클래스

│ ├── services/ ← 외부 연동 (ex. Elasticsearch)

│ │ └── logger.py ← log\_request 함수 등

│ └── config.py ← 설정 (ex. API Endpoint, CORS 등)

├── requirements.txt

└── .env

**✅ main.py 예시 (루트에 두되 app/ 아래 모듈 임포트)**

python

복사편집

from fastapi import FastAPI

from app.api.proxy import router as proxy\_router

app = FastAPI(title="AI Proxy Gateway", version="1.0.0")

app.include\_router(proxy\_router)

@app.get("/")

def root():

return {"message": "AI Proxy Gateway is running"}

**✅ 예: app/api/proxy.py**

python

복사편집

from fastapi import APIRouter, Request, HTTPException

from app.core.filter import SensitiveDataFilter

from app.core.validator import InputValidator

from app.services.logger import log\_request

import httpx, os

from typing import Dict, Any

router = APIRouter(prefix="/proxy", tags=["proxy"])

AI\_ENDPOINTS = {

"openai": "https://api.openai.com/v1",

"anthropic": "https://api.anthropic.com/v1"

}

sensitive\_filter = SensitiveDataFilter()

input\_validator = InputValidator()

@router.post("/{provider}/{endpoint}")

async def proxy(provider: str, endpoint: str, request: Request):

...

**✅ 장점 요약**

| **기존 구조** | **새 구조** |
| --- | --- |
| 루트에 모든 파일 혼재 | 모듈별로 기능 정리 (api, core, service) |
| app/ 하위에 security만 있어서 import 충돌 위험 | core/, api/, services/로 분리 |
| 테스트 및 확장 어려움 | 라우터 분리 → test 가능, 유지보수 용이 |

**🛠️ 실행 명령어 (변함없음)**

bash

복사편집

uvicorn main:app --reload --host 0.0.0.0 --port 8000

**📌 원하시면**

* main.py, router, filter.py, validator.py 샘플 파일 전체 제공 가능
* 기존 코드 자동 구조화 스크립트 제공 가능

대표님께서 원하신다면,  
📂 현재의 AiGov-Proxy/ 구조를 캡처해서 보여주시거나  
💡 위 구조로 리팩터링된 기본 파일 세트를 만들어드릴 수 있습니다.

어떤 방식으로 진행하실까요? (자동 정리, 샘플 코드 생성 등)

**나의 말:**

우툰부 환경에서 현재 폴더 구조를 확인하는 명령어를 알려줘...

**ChatGPT의 말:**

대표님, WSL(Ubuntu) 환경에서 현재 폴더(디렉토리) 구조를 계층적으로 확인하려면 아래 명령어를 사용하시면 됩니다:

**✅ 현재 디렉토리 트리 구조 확인 명령어**

bash

복사편집

tree -L 2

| **요소** | **설명** |
| --- | --- |
| tree | 폴더 구조를 트리 형태로 출력하는 명령 |
| -L 2 | 하위 2단계까지 출력 (필요 시 3, 4 등으로 조정 가능) |

**📦 예시 출력**

bash

복사편집

.

├── main.py

├── app

│ ├── api

│ ├── core

│ ├── services

│ └── \_\_init\_\_.py

├── requirements.txt

└── .env

**📌 설치가 안 되어 있다면?**

Ubuntu에서 tree가 기본 설치되어 있지 않을 수 있습니다. 아래 명령어로 먼저 설치하세요:

bash

복사편집

sudo apt update

sudo apt install tree

**🧠 대안 명령어 (설치 없이 사용 가능)**

bash

복사편집

ls -R

또는 계층적이진 않지만 디렉토리 구조를 빠르게 확인:

bash

복사편집

find . -type d

필요하시면 tree 명령어 출력 결과를 기준으로 폴더 리팩터링 구조를 자동 제안해드릴 수도 있습니다.  
설치 완료 후 출력 결과를 공유해주시면 함께 점검해드릴게요!

**나의 말:**

현재 트리 구조야. 확인 후 위의 권장 FastAPI 프로젝트 구조와 비교해서 가이드 해줘 jake@ChihoKim:/mnt/c/Users/김치호/OneDrive/바탕 화면/cursor/AiGov/AiGov-Proxy$ tree -L 4 . ├── Dockerfile ├── \_\_pycache\_\_ │ └── main.cpython-310.pyc ├── app │ ├── filters.py │ └── security │ ├── filter.py │ └── validator.py ├── docker-compose.yml ├── main.py ├── rebuff │ ├── LICENSE │ ├── Makefile │ ├── README.md │ ├── detection.puml │ ├── docs │ │ ├── README.md │ │ ├── SUMMARY.md │ │ ├── how-it-works.md │ │ ├── quickstart.md │ │ └── self-hosting.md │ ├── javascript-sdk │ │ ├── README.md │ │ ├── package.json │ │ ├── src │ │ │ ├── api.ts │ │ │ ├── config.ts │ │ │ ├── index.ts │ │ │ ├── interface.ts │ │ │ ├── lib │ │ │ ├── sdk.ts │ │ │ └── tactics │ │ ├── tests │ │ │ ├── helpers.ts │ │ │ ├── index.test.ts │ │ │ ├── insert-chroma-vectors.ts │ │ │ ├── setup-and-run-tests.sh │ │ │ └── wait-for-chroma.ts │ │ ├── tsconfig.json │ │ └── yarn.lock │ ├── python-sdk │ │ ├── Makefile │ │ ├── README.md │ │ ├── poetry.lock │ │ ├── pyproject.toml │ │ ├── python-sdk-examples.ipynb │ │ ├── rebuff │ │ │ ├── \_\_init\_\_.py │ │ │ ├── \_version.py │ │ │ ├── detect\_pi\_heuristics.py │ │ │ ├── detect\_pi\_openai.py │ │ │ ├── detect\_pi\_vectorbase.py │ │ │ ├── rebuff.py │ │ │ └── sdk.py │ │ └── tests │ │ ├── \_\_init\_\_.py │ │ ├── conftest.py │ │ ├── test\_integration.py │ │ ├── test\_langchain.py │ │ ├── test\_sdk.py │ │ └── utils.py │ ├── server │ │ ├── README.md │ │ ├── components │ │ │ ├── ApikeyDisplay.tsx │ │ │ ├── AppContext.tsx │ │ │ ├── CodeSamples.tsx │ │ │ ├── Layout.tsx │ │ │ ├── LoginButtonWithInstructions.tsx │ │ │ ├── Navbar.tsx │ │ │ ├── ProfileSettings.tsx │ │ │ ├── PromptHistory.tsx │ │ │ ├── PromptInjectionStats.tsx │ │ │ ├── Section.tsx │ │ │ ├── SequenceDiagram.tsx │ │ │ └── SocialIcons.tsx │ │ ├── lib │ │ │ ├── account-helpers.ts │ │ │ ├── custom-error.ts │ │ │ ├── detect-helpers.ts │ │ │ ├── general-helpers.ts │ │ │ ├── openai.ts │ │ │ ├── pinecone-client.ts │ │ │ ├── rebuff-api.ts │ │ │ ├── rebuff.ts │ │ │ ├── schema.ts │ │ │ ├── supabase.ts │ │ │ └── templates.ts │ │ ├── netlify.toml │ │ ├── next-env.d.ts │ │ ├── next.config.js │ │ ├── package-lock.json │ │ ├── package.json │ │ ├── pages │ │ │ ├── \_app.tsx │ │ │ ├── \_document.tsx │ │ │ ├── api │ │ │ ├── docs │ │ │ └── index.tsx │ │ ├── postcss.config.js │ │ ├── public │ │ │ ├── android-chrome-192x192.png │ │ │ ├── android-chrome-512x512.png │ │ │ ├── apple-touch-icon.png │ │ │ ├── favicon-16x16.png │ │ │ ├── favicon-32x32.png │ │ │ ├── favicon.ico │ │ │ ├── logo.png │ │ │ └── site.webmanifest │ │ ├── requirements.txt │ │ ├── sql\_setup │ │ │ ├── functions │ │ │ └── tables │ │ ├── styles │ │ │ ├── app.css │ │ │ └── tailwind.css │ │ ├── tailwind.config.js │ │ ├── tsconfig.json │ │ ├── tsconfig.tsbuildinfo │ │ ├── types │ │ │ └── types.d.ts │ │ ├── utils │ │ │ └── apikeys.ts │ │ └── venv │ │ ├── bin │ │ ├── include │ │ ├── lib │ │ ├── lib64 -> lib │ │ └── pyvenv.cfg │ └── venv │ ├── bin │ │ ├── python -> python3 │ │ ├── python3 -> /usr/bin/python3 │ │ └── python3.10 -> python3 │ ├── include │ ├── lib │ │ └── python3.10 │ ├── lib64 -> lib │ └── pyvenv.cfg └── venv ├── bin │ ├── Activate.ps1 │ ├── activate │ ├── activate.csh │ ├── activate.fish │ ├── dotenv │ ├── fastapi │ ├── normalizer │ ├── pip │ ├── pip3 │ ├── pip3.10 │ ├── python -> python3 │ ├── python3 -> /usr/bin/python3 │ ├── python3.10 -> python3 │ └── uvicorn ├── include ├── lib │ └── python3.10 │ └── site-packages ├── lib64 -> lib └── pyvenv.cfg

**ChatGPT의 말:**

대표님, 공유해주신 트리 구조를 확인해보니 현재 AiGov-Proxy 프로젝트는 비교적 잘 정리되어 있지만,  
**FastAPI 애플리케이션 구조 관점에서는 개선의 여지가 있습니다.**  
아래에 문제점과 함께 **권장 구조**로 정리해드리겠습니다.

**❗ 현재 구조의 문제점 요약**

| **문제 구간** | **문제점** |
| --- | --- |
| main.py가 루트에 있음 | 파일이 많아질수록 유지보수 어려움 |
| app/filters.py와 app/security/가 혼재 | 역할별 모듈이 명확히 나뉘지 않음 |
| rebuff/는 외부 오픈소스라 내부 구조와 혼동 가능 | 내부 코드와 외부 코드가 혼재되어 혼란 초래 |
| venv 디렉토리가 중복으로 존재 | 실제 사용하는 환경이 명확하지 않음 |

**✅ 권장 구조 (리팩토링안)**

bash

복사편집

AiGov-Proxy/

├── app/ ← 모든 FastAPI 애플리케이션 코드

│ ├── \_\_init\_\_.py

│ ├── main.py ← 앱 진입점: uvicorn app.main:app

│ ├── api/ ← API 라우팅

│ │ └── proxy.py

│ ├── core/ ← 비즈니스 로직 (필터, 검증기)

│ │ ├── filter.py

│ │ └── validator.py

│ ├── services/ ← 로그, 외부 API 연동

│ │ └── logger.py

│ ├── config.py ← 설정값 (CORS, endpoint 등)

│ └── models/ ← 추후 DB나 Pydantic 모델 정의용

├── Dockerfile

├── docker-compose.yml

├── requirements.txt

├── .env

├── rebuff/ ← 외부 오픈소스 디렉토리 (건드리지 않음)

└── venv/ ← 1개의 가상환경만 유지

**🔁 변경 작업 안내**

**🔹 1. main.py → app/main.py로 이동**

* 루트에 있던 main.py를 app/ 폴더 안으로 옮깁니다.
* 실행은 다음처럼:

bash

복사편집

uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

**🔹 2. app/security/ → app/core/로 이동 + 통합**

* filter.py, validator.py는 app/core/로 옮깁니다.
* 필요 시 core/filter.py, core/validator.py로 구성

**🔹 3. API 라우팅 분리 (선택 사항)**

* proxy\_request() 등은 app/api/proxy.py로 이동
* app/main.py에서는 아래처럼 include\_router() 사용:

python

복사편집

from fastapi import FastAPI

from app.api.proxy import router as proxy\_router

app = FastAPI()

app.include\_router(proxy\_router)

**🔹 4. Rebuff 디렉토리**

* rebuff/는 건드리지 않고 3rdparty 또는 external/로 따로 분리해도 좋습니다.
* 커스터마이징 없이 참조만 한다면 명확히 구분하세요.

**✅ 실행 환경 통합 권장**

* 현재 rebuff/server/venv, AiGov-Proxy/venv, 루트의 venv가 중복 존재
* 명확히 **하나만 사용** (예: AiGov-Proxy/venv)
* 불필요한 가상환경은 rm -rf venv로 제거 가능

**✨ 리팩토링 후의 실행 방법**

1. 진입점은 항상 app/main.py 안의 app = FastAPI() 기준
2. 실행 명령어는:

bash

복사편집

uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

1. 접속은: <http://localhost:8000/docs>

**🔧 원하시면**

* 제가 직접 리팩토링된 디렉토리 구조 + 파일 샘플 제공
* 기존 main.py와 filter.py, validator.py 자동 이전

이대로 정리해드릴까요?  
아니면 대표님이 직접 이동하신 뒤 제가 main.py 코드를 수정해드릴까요?

**나의 말:**

filter.py가 맞아 filters.py가 맞아?

**ChatGPT의 말:**

아주 좋은 질문입니다, 대표님.

**✅ 결론부터 말씀드리면:**

일반적으로 **filter.py** 가 더 권장됩니다.  
Python 모듈은 **단수 명칭**으로 작성하는 것이 관례적이고 가독성 면에서도 좋습니다.

**📌 이유**

| **비교** | **filter.py** | **filters.py** |
| --- | --- | --- |
| 의미 | 필터 기능을 담은 **하나의 모듈** | 여러 필터가 들어 있는 듯한 **불명확한 추상화** |
| Python 스타일 가이드 (PEP8) | ✅ 단수형 모듈 이름 권장 | 🔸 복수형은 피함 |
| 관례 | config.py, validator.py, model.py 등 대부분 단수 사용 | configs.py, validators.py는 특정 상황에서만 |

특히 클래스 이름이 SensitiveDataFilter, InputValidator라면  
해당 정의를 담는 파일도 filter.py, validator.py로 단수형이 더 자연스럽습니다.

**✅ 권장 구조 예시**

pgsql

복사편집

app/

├── core/

│ ├── filter.py ← SensitiveDataFilter

│ └── validator.py ← InputValidator

그리고 main.py나 API 모듈에서는:

python

복사편집

from app.core.filter import SensitiveDataFilter

from app.core.validator import InputValidator

이처럼 명확하게 읽히고 import 경로도 간결합니다.

**예외: filters.py를 사용하는 경우**

* 여러 종류의 **분리된 필터**(예: EmailFilter, PasswordFilter 등)를 담고 있을 경우엔 filters.py도 허용됩니다.
* 하지만 그럴 때도 보통은 **디렉터리로 나누고** 각 필터를 별도 파일로 관리합니다:

markdown

복사편집

core/

└── filters/

├── \_\_init\_\_.py

├── email.py

├── password.py

**✅ 최종 정리**

| **상황** | **추천 파일명** |
| --- | --- |
| 필터 클래스가 하나 또는 단일 기능 중심 | filter.py ✅ |
| 여러 필터 기능이 모듈화되어 있을 경우 | filters/ 디렉터리 사용 🔄 |

현재 구조에서는 **filter.py가 더 명확하고 PEP8에도 부합**합니다.  
필요하시면 해당 모듈명 변경에 따른 import 정리도 도와드릴게요. 계속 이어가시겠어요?

**나의 말:**

권장 구조에서 app 하위에 폴더는 생성했는데, \_\_init\_\_.py 및 config.py파일, api/proxy.py 파일, services/logger.py이 없어

**ChatGPT의 말:**

훌륭하십니다, 대표님.  
app/ 하위 디렉터리 구조를 잘 만드셨다면, 이제 필요한 파일들을 채워 넣기만 하면 됩니다.  
제가 바로 작성해드릴게요.

**✅ 1. app/\_\_init\_\_.py (빈 파일)**

이 파일은 해당 디렉터리를 **Python 패키지로 인식**시키기 위한 용도입니다.

python

복사편집

# app/\_\_init\_\_.py

(내용 없어도 됩니다)

**✅ 2. app/config.py**

python

복사편집

# app/config.py

AI\_ENDPOINTS = {

"openai": "https://api.openai.com/v1",

"anthropic": "https://api.anthropic.com/v1"

}

CORS\_ALLOW\_ORIGINS = ["\*"] # 개발용. 운영 시 도메인 제한 권장

**✅ 3. app/services/logger.py**

python

복사편집

# app/services/logger.py

from datetime import datetime

from elasticsearch import Elasticsearch

from loguru import logger

import os

from typing import Dict, Any

es\_client = Elasticsearch(

os.getenv("ELASTICSEARCH\_URL", "http://localhost:9200") # 기본 포트로 수정

)

async def log\_request(

user\_id: str,

provider: str,

endpoint: str,

request\_data: Dict[str, Any],

response\_data: Dict[str, Any],

validation\_result: Dict[str, Any]

):

log\_entry = {

"timestamp": datetime.utcnow(),

"user\_id": user\_id,

"provider": provider,

"endpoint": endpoint,

"request\_data": request\_data,

"response\_data": response\_data,

"validation\_result": validation\_result

}

try:

await es\_client.index(index="ai-proxy-logs", document=log\_entry)

except Exception as e:

logger.error(f"Elasticsearch logging failed: {str(e)}")

**✅ 4. app/api/proxy.py**

python

복사편집

# app/api/proxy.py

from fastapi import APIRouter, Request, HTTPException

from app.core.filter import SensitiveDataFilter

from app.core.validator import InputValidator

from app.services.logger import log\_request

from app.config import AI\_ENDPOINTS

import httpx

router = APIRouter(prefix="/proxy", tags=["AI Proxy"])

sensitive\_filter = SensitiveDataFilter()

input\_validator = InputValidator()

@router.post("/{provider}/{endpoint}")

async def proxy\_request(provider: str, endpoint: str, request: Request):

if provider not in AI\_ENDPOINTS:

raise HTTPException(status\_code=400, detail=f"Unsupported provider: {provider}")

body = await request.json()

validation\_result = await input\_validator.validate\_request(body)

if not validation\_result["is\_valid"]:

raise HTTPException(status\_code=400, detail=validation\_result)

filtered\_body = sensitive\_filter.filter\_request(body)

try:

async with httpx.AsyncClient() as client:

response = await client.post(

f"{AI\_ENDPOINTS[provider]}/{endpoint}",

json=filtered\_body,

headers={"Authorization": request.headers.get("Authorization", "")}

)

response\_data = response.json()

filtered\_response = sensitive\_filter.filter\_response(response\_data)

user\_id = request.headers.get("X-User-ID", "anonymous")

await log\_request(user\_id, provider, endpoint, filtered\_body, filtered\_response, validation\_result)

return filtered\_response

except Exception as e:

raise HTTPException(status\_code=500, detail="Proxy failed: " + str(e))

**✅ main.py (참고용)**

루트가 아닌 app/main.py 로 옮겨졌다면:

python

복사편집

# app/main.py

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from app.api.proxy import router as proxy\_router

from app.config import CORS\_ALLOW\_ORIGINS

app = FastAPI(title="AI Proxy Gateway", version="1.0.0")

app.add\_middleware(

CORSMiddleware,

allow\_origins=CORS\_ALLOW\_ORIGINS,

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"]

)

app.include\_router(proxy\_router)

@app.get("/")

def root():

return {"message": "AI Proxy Gateway is running"}

**✅ 실행 명령어**

bash

복사편집

uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

이제 폴더와 파일이 완전히 갖춰졌습니다.  
이 상태에서 실행 후 /docs가 정상 작동하는지 확인하신 뒤,  
필요하면 filter.py, validator.py 내용도 정리해드리겠습니다.

진행해보시고 결과 알려주세요!

**나의 말:**

폴더 구조 설정 및 파일 생성은 우분투 환경에서 실행했고, 이후는 파이선 가상환경(python3 -m venv venv, source...)에서 실행할려고 하는데 맞아?

**ChatGPT의 말:**

네, 정확하게 잘 이해하고 계십니다, 대표님!  
지금까지 설정하신 구조에서 이후 작업은 **Python 가상환경(venv)** 내에서 실행하시는 것이 맞습니다.

**✅ 전체 흐름 요약**

**1️⃣ WSL(Ubuntu) 환경에서 작업 디렉토리 이동**

bash

복사편집

cd ~/AiGov/AiGov-Proxy

**2️⃣ 가상환경 생성 (최초 1회)**

bash

복사편집

python3 -m venv venv

결과: ./venv/ 폴더가 생성됨

**3️⃣ 가상환경 활성화**

bash

복사편집

source venv/bin/activate

프롬프트가 아래처럼 (venv)로 바뀌면 성공:

ruby

복사편집

(venv) jake@ChihoKim:~/AiGov/AiGov-Proxy$

**4️⃣ 의존 패키지 설치 (최초 또는 변경 시마다)**

bash

복사편집

pip install -r requirements.txt

requirements.txt에 다음 항목들이 포함되어 있어야 함:

txt

복사편집

fastapi

uvicorn

httpx

python-dotenv

loguru

elasticsearch

(guardrails, langchain 등은 필요 시 추가)

**5️⃣ FastAPI 서버 실행**

bash

복사편집

uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

→ 브라우저에서:

bash

복사편집

http://localhost:8000/docs

또는

arduino

복사편집

http://<WSL IP>:8000/docs

**✅ 가상환경 내에서만 실행해야 하는 이유**

| **이유** | **설명** |
| --- | --- |
| 독립성 확보 | 시스템 파이썬에 영향 주지 않음 |
| 패키지 충돌 방지 | Rebuff, Guardrails, Langchain 등 최신 라이브러리 관리 |
| 버전 제어 | 프로젝트마다 다른 버전 사용 가능 |

**🧠 팁**

가상환경을 나가고 싶을 땐:

bash

복사편집

deactivate

**✅ 요약**

| **단계** | **명령어** |
| --- | --- |
| 가상환경 생성 | python3 -m venv venv |
| 가상환경 진입 | source venv/bin/activate |
| 패키지 설치 | pip install -r requirements.txt |
| FastAPI 실행 | uvicorn app.main:app --reload --host 0.0.0.0 --port 8000 |