# AI Python Developer Assessment — Full Solution

This document provides a complete, implementation-ready solution to the assessment. It covers API design, an AI compliance agent, user interaction for auto-correction, testing, validation, performance optimizations, risks, and delivery guidelines.

## 1. Project Overview

Build a service that accepts PDF/Word files, extracts text, evaluates compliance with English writing guidelines (grammar, clarity, structure), returns a detailed report, and optionally returns an auto-corrected document.

## 2. High Level Architecture

The system is composed of: (1) a FastAPI service for uploads and interaction, (2) a text extraction layer for PDF/Word, (3) an AI Compliance Agent that combines rules + NLP checks, and (4) a modification pipeline that rewrites text while preserving basic formatting.

text

Components

Client (Web/UI or API consumer)

HTTPS JSON + multipart/form-data

FastAPI Service

/health/analyze (upload) . /modify (apply fixes)

Auth (token) . Rate limit . Validation

Extraction

PDF: pdfminer.six

DOCX: python-docx

Fallback OCR (optional)

AI Compliance Agent

Grammar: LanguageTool

Style: heuristics

Readability metrics

. spaCy for parsing

. GPT (optional)

Storage (temp, S3, etc.)

Encrypted at rest

Auto-cleanup

Modifier / Rewriter

Rule-based rewrite

GPT for rewrite

## 3. API Design (FastAPI)

Endpoints are intentionally minimal and easy to integrate. Use bearer-token auth in production.

| **Method** | **Path** | **Purpose** | **Payload / Response** |  |
| --- | --- | --- | --- | --- |
| GET | /health | Health check | 200 OK |  |
| POST | /analyze | Upload a PDF/DOCX and get compliance report | multipart/form-data (file); JSON report |  |
| POST | /modify | Return guideline-compliant version | JSON (doc\_id | text, rules); returns file or text |

### FastAPI Reference Implementation (core app)

python

*# app/main.py*

from fastapi import FastAPI, UploadFile, File, HTTPException

from fastapi.responses import JSONResponse, FileResponse

from pydantic import BaseModel

from typing import Optional, List

import uuid, os

from .extract import extract\_text

from .agent import analyze\_text, modify\_text

from .storage import save\_temp, get\_temp\_path

app = FastAPI(title="Compliance API")

class ModifyRequest(BaseModel):

doc\_id: Optional[str] = None

text: Optional[str] = None

rules: Optional[List[str]] = None

format: str = "docx" *# or "txt" / "pdf"*

@app.get("/health")

def health():

return {'status': 'ok'}

@app.post("/analyze")

async def analyze(file: UploadFile = File(...)):

if file.content\_type not in {"application/pdf",

"application/vnd.openxmlformats-officedocument.wordprocessingml.document"}:

raise HTTPException(status\_code=400, detail="Only PDF or DOCX accepted")

doc\_id = str(uuid.uuid4())

disk\_path = save\_temp(doc\_id, file)

text = extract\_text(disk\_path, content\_type=file.content\_type)

report = analyze\_text(text)

return JSONResponse({'doc\_id': doc\_id, 'report': report})

@app.post("/modify")

async def modify(req: ModifyRequest):

if not req.doc\_id and not req.text:

raise HTTPException(status\_code=400, detail="Provide doc\_id or text")

text = req.text

if req.doc\_id and not text:

text = extract\_text(get\_temp\_path(req.doc\_id))

updated\_text, changes = modify\_text(text, rules=req.rules or [])

out\_path = get\_temp\_path(f"{uuid.uuid4()}.{req.format}")

if req.format == "docx":

from docx import Document

doc = Document()

for para in updated\_text.split('\n'):

doc.add\_paragraph(para)

doc.save(out\_path)

else:

with open(out\_path, "w", encoding="utf-8") as f:

f.write(updated\_text)

return FileResponse(out\_path, filename=os.path.basename(out\_path))

### Extraction Layer

python

*# app/extract.py*

from typing import Optional

from pdfminer.high\_level import extract\_text as pdf\_extract

from docx import Document

def extract\_text(path: str, content\_type: Optional[str] = None) -> str:

if content\_type == "application/pdf" or path.lower().endswith(".pdf"):

return pdf\_extract(path)

if content\_type == "application/vnd.openxmlformats-officedocument.wordprocessingml.document" or path.lower().endswith(".docx"):

doc = Document(path)

return "\n".join(p.text for p in doc.paragraphs)

raise ValueError("Unsupported file type")

### Storage Helpers

python

*# app/storage.py*

import os, shutil, tempfile

ROOT = tempfile.gettempdir()

NAMESPACE = "compliance\_api"

def namespace\_dir():

path = os.path.join(ROOT, NAMESPACE)

os.makedirs(path, exist\_ok=True)

return path

def save\_temp(key: str, upload\_file) -> str:

path = os.path.join(namespace\_dir(), key)

filename = getattr(upload\_file, "filename", None) or "upload.bin"

ext = os.path.splitext(filename)[1]

full = path + ext

with open(full, "wb") as f:

shutil.copyfileobj(upload\_file.file, f)

return full

def get\_temp\_path(key: str) -> str:

p = os.path.join(namespace\_dir(), key)

if os.path.exists(p):

return p

for fn in os.listdir(namespace\_dir()):

if fn.startswith(key):

return os.path.join(namespace\_dir(), fn)

raise FileNotFoundError(key)

## 4. AI Compliance Agent

The agent combines deterministic rules with NLP checks for better precision/recall. It yields a structured report with categories: Grammar, Spelling, Punctuation, Style, Readability, Structure.

### Rules & Metrics

* Grammar & Spelling: LanguageTool (language\_tool\_python)
* Tokenization/Parsing: spaCy (en\_core\_web\_sm)
* Readability: Flesch Reading Ease, Flesch-Kincaid Grade
* Style Heuristics: passive voice ratio, sentence length limits, repeated words, banned phrases (e.g., 'basically', 'very')
* Structure: title present, heading ratio, paragraph length within bounds, bullet usage when lists are detected

### Report JSON Schema (Example)

json

{

"summary": {"overall\_compliance": 0.86, "grade": "B"},

"metrics": {"readability": {"flesch": 61.2, "fk\_grade": 8.4},

"style": {"passive\_ratio": 0.09, "avg\_sentence\_len": 17.2}},

"violations": [{"id":"G001","type":"grammar","message":"Subject-verb agreement","span":"...","suggestion":"..."}]

}

### Agent Implementation (reference)

python

*# app/agent.py*

import language\_tool\_python

import textstat

import re

from typing import List, Tuple, Dict

import spacy

tool = language\_tool\_python.LanguageToolPublicAPI('en-US')

nlp = spacy.load('en\_core\_web\_sm')

BANNED\_PHRASES = {'very', 'basically', 'literally', 'obviously'}

def analyze\_text(text: str) -> Dict:

doc = nlp(text)

sentences = list(doc.sents)

passive\_cnt = sum(1 for s in sentences if has\_passive(s))

avg\_len = sum(len(s.text.split()) for s in sentences) / max(1, len(sentences))

flesch = textstat.flesch\_reading\_ease(text)

fk = textstat.flesch\_kincaid\_grade(text)

matches = tool.check(text)

violations = []

for m in matches:

violations.append({

'id': m.ruleId,

'type': m.category,

'message': m.message,

'span': text[m.offset:m.offset+m.errorLength],

'suggestion': (m.replacements or [""])[0]

})

*# Style heuristics*

for phrase in BANNED\_PHRASES:

for hit in re.finditer(rf"\b({re.escape(phrase)})\b", text, flags=re.I):

violations.append({

'id': "STYLE\_BANNED",

'type': "style",

'message': f"Discouraged word: '{phrase}'",

'span': phrase,

'suggestion': "Remove or replace with a stronger term."

})

overall = score\_overall(len(violations), passive\_cnt, avg\_len, flesch, fk)

return {

"summary": {"overall\_compliance": overall, "grade": letter\_grade(overall)},

"metrics": {

"readability": {'flesch': flesch, "fk\_grade": fk},

"style": {'passive\_ratio': passive\_cnt/max(1,len(sentences)), 'avg\_sentence\_len': avg\_len},

},

"violations": violations

}

def has\_passive(span) -> bool:

lemmas = [t.lemma\_.lower() for t in span]

return any(t.tag\_ == "VBN" for t in span) and any(l in {"be"} for l in lemmas)

def score\_overall(num, passive, avg\_len, flesch, fk):

p = max(0, 1 - (num/100)) \* 0.5

s = max(0, 1 - min(passive/10, 1)) \* 0.2

r = min(max((flesch/100), 0), 1) \* 0.3

return round(p + s + r, 2)

def letter\_grade(score):

return "A" if score >= 0.9 else "B" if score >= 0.8 else "C" if score >= 0.7 else "D" if score >= 0.6 else "F"

def modify\_text(text: str, rules: List[str]) -> Tuple[str, List[Dict]]:

matches = tool.check(text)

changes = []

out = text

offset = 0

for m in matches:

if m.replacements:

start, end = m.offset, m.offset + m.errorLength

suggestion = m.replacements[0]

changes.append({'from': out[start+offset:end+offset], "to": suggestion, "pos": [start+offset, end+offset]})

out = out[:start+offset] + suggestion + out[end+offset:]

offset += len(suggestion) - (end - start)

for phrase in BANNED\_PHRASES:

out = re.sub(rf"\b({re.escape(phrase)})\b", "", out, flags=re.I)

return out, changes

## 5. User Interaction & Compliance Correction

The client first calls /analyze to receive a report and doc\_id. It can display the violations grouped by category. If the user clicks 'Auto-fix', the client calls /modify with doc\_id (and optional rules) and downloads the corrected file.

## 6. Testing & Validation

### Unit Tests (pytest)

python

*# tests/test\_agent.py*

from app.agent import analyze\_text, modify\_text

def test\_analyze\_basic():

report = analyze\_text("This are bad sentence. It was written by me.")

assert "summary" in report and "violations" in report

assert isinstance(report["violations"], list)

def test\_modify\_text():

updated, changes = modify\_text("He are good.", [])

assert isinstance(updated, str)

### Integration Tests (FastAPI TestClient)

python

*# tests/test\_api.py*

from fastapi.testclient import TestClient

from app.main import app

client = TestClient(app)

def test\_health():

r = client.get("/health")

assert r.status\_code == 200

def test\_analyze\_rejects\_unknown():

files = {"file": ("a.txt", b"hello", "text/plain")}

r = client.post("/analyze", files=files)

assert r.status\_code == 400

## 7. Performance Optimization

* Stream uploads to disk; avoid loading entire files into memory
* Chunked extraction for long PDFs
* Cache spaCy model and LanguageTool instance as singletons
* Use background tasks or Celery for very large documents
* Add pagination for long reports
* Preheat models on startup to remove cold-start latency

## 8. Security Considerations

* Validate MIME types and file signatures
* Limit file size and page count
* Store files in a non-executable temp directory with TTL cleanup
* Sanitize text before logging
* Add AuthN/Z (bearer tokens)
* Rate limiting and request quotas

## 9. Deliverables

* FastAPI project with endpoints and reference implementation
* AI agent with rule + NLP checks
* Tests (unit + integration) and sample documents
* README with setup instructions

## 10. Example Project Structure

text

ai-compliance/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py

│ ├── extract.py

│ ├── agent.py

│ └── storage.py

├── tests/

│ ├── test\_agent.py

│ └── test\_api.py

├── requirements.txt

├── README.md

└── pyproject.toml

## 11. Requirements (pip)

text

fastapi

uvicorn[standard]

python-docx

pdfminer.six

language-tool-python

spacy

textstat

pytest

## 12. README — Run Locally

bash

python -m venv .venv && source .venv/bin/activate

pip install -r requirements.txt

python -m spacy download en\_core\_web\_sm

uvicorn app.main:app --reload

*# Analyze*

curl -F "file=@samples/sample.pdf" http://localhost:8000/analyze

*# Modify (download corrected docx)*

curl -X POST http://localhost:8000/modify -H "Content-Type: application/json" \

-d '{"doc\_id":"<from\_analyze>", "format":"docx"}' --output corrected.docx

## 13. Risk Assessment & Mitigations

* Accuracy: combine deterministic rules + model; allow human in the loop review
* Performance: streaming, caching, async, background jobs
* Robustness: fallback OCR for scanned PDFs; clear error messages; timeouts and circuit breakers

## 14. Suggested Timeline

Day 1: Scaffold FastAPI, extraction, basic agent; Day 2: reports, modify pipeline, tests; Day 3: polish, docs, sample datasets, CI.

## 15. Conclusion

This solution is production-oriented yet lightweight. It demonstrates end-to-end handling of documents, AI-driven compliance analysis, and automated corrections with clear APIs and tests.