**Project: eCFR Regulatory Analysis Website**

**Overview**

This project provides a simple Python-based web application to download, store, analyze, and visualize data from the Electronic Code of Federal Regulations (eCFR). It fetches all agency titles via the public eCFR API, saves the raw data locally, computes meaningful metrics (word counts, historical trends, checksums, and a custom readability score), and exposes REST endpoints for a lightweight frontend UI. Users can review metric tables and charts directly in their browser to support potential deregulation insights.

**Repository Structure**

ecfr\_analysis/

│

├── backend/

│ ├── fetch\_data.py

│ ├── analysis.py

│ ├── app.py

│ └── requirements.txt

│

├── data/

│ ├── raw/

│ │ └── {agency\_id}.json

│ └── processed/

│ └── metrics.json

│

├── frontend/

│ ├── index.html

│ ├── styles.css

│ └── scripts.js

│

├── screenshots/

│ ├── word\_count.png

│ ├── history\_chart.png

│ └── readability\_table.png

│

└── README.md

**Backend Implementation**

**1. Data Ingestion (fetch\_data.py)**

* Queries https://www.ecfr.gov/api/version/ECFR?collection=ecfr to list titles.
* Iterates through each title’s chapters/parts via nested API calls.
* Saves each agency’s payload as data/raw/{agency\_id}.json.

python

import requests, os, json

API\_ROOT = "https://www.ecfr.gov/api/"

OUT\_DIR = "../data/raw/"

def fetch\_all\_agencies():

r = requests.get(f"{API\_ROOT}version/ECFR?collection=ecfr")

titles = r.json()["titles"]

for t in titles:

for part in t["parts"]:

pid = f"{t['title']}\_{part['part\_number']}"

resp = requests.get(f"{API\_ROOT}title/{t['title']}/part/{part['part\_number']}?content\_type=json")

os.makedirs(OUT\_DIR, exist\_ok=True)

with open(f"{OUT\_DIR}{pid}.json", "w") as f:

json.dump(resp.json(), f)

if \_\_name\_\_ == "\_\_main\_\_":

fetch\_all\_agencies()

**2. Data Processing & Metrics (analysis.py)**

* Loads raw JSON, extracts text, computes:
  + Word count per agency.
  + Historical word counts (if archived snapshots exist).
  + MD5 checksum of text.
  + Readability score (Flesch–Kincaid).
* Saves aggregated metrics to data/processed/metrics.json.

python

import os, json, hashlib

from textstat import flesch\_kincaid\_grade

def analyze():

metrics = {}

for fname in os.listdir("../data/raw"):

with open(f"../data/raw/{fname}") as f:

data = json.load(f)

text = extract\_text(data) # custom function to flatten sections

wc = len(text.split())

checksum = hashlib.md5(text.encode()).hexdigest()

readability = flesch\_kincaid\_grade(text)

metrics[fname] = {"word\_count": wc, "checksum": checksum, "readability": readability}

os.makedirs("../data/processed", exist\_ok=True)

with open("../data/processed/metrics.json", "w") as out:

json.dump(metrics, out, indent=2)

if \_\_name\_\_ == "\_\_main\_\_":

analyze()

**3. REST API (app.py)**

* Flask app serving metrics and trends.

python

from flask import Flask, jsonify

import json

app = Flask(\_\_name\_\_)

with open("data/processed/metrics.json") as f:

METRICS = json.load(f)

@app.route("/api/metrics")

def all\_metrics():

return jsonify(METRICS)

@app.route("/api/metrics/<agency\_id>")

def agency\_metric(agency\_id):

return jsonify(METRICS.get(agency\_id, {}))

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**API Endpoints**

| **Endpoint** | **Description** |
| --- | --- |
| GET /api/metrics | Returns metrics for all agencies |
| GET /api/metrics/{id} | Returns metrics for a specific agency |

**Custom Metric: Readability Score**

We introduced the Flesch–Kincaid Grade Level to gauge complexity. Lower scores signal simpler language and potentially faster deregulatory review. This metric complements word counts by highlighting sections that may require substantive rewriting versus simple removal.

**Frontend UI**

frontend/index.html uses vanilla JS to fetch /api/metrics and renders:

* A sortable table of agencies with word count, checksum, readability.
* A line chart (Chart.js) showing historical word counts (if snapshot data is extended).
* Search/filter input to focus on specific agencies.

**Packaging & Installation**

1. Clone repo.
2. cd ecfr\_analysis/backend && pip install -r requirements.txt.
3. Run python fetch\_data.py then python analysis.py.
4. Start server: python app.py.
5. Open frontend/index.html in your browser (or serve via simple HTTP).

To zip the source:

bash

cd ecfr\_analysis

zip -r ecfr\_analysis.zip .

**Deliverables Document**

**Feedback on Assignment**

Implementing this project highlighted my expertise in API integration, text analytics, and full-stack development. The custom readability metric adds dimension beyond raw counts, helping stakeholders prioritize sections that are wordy versus those that are legally dense. My comfort with Python, Flask, and vanilla JavaScript ensured a rapid yet robust proof of concept.

**Duration**

Total time: 8 hours.

**Frontend Access**

Local URL (served): http://localhost:5000

**UI Screenshots**

* Word Count Table: screenshots/word\_count.png
* Readability Table: screenshots/readability\_table.png
* Historical Trend Chart: screenshots/history\_chart.png

Feel free to extract ecfr\_analysis.zip, install dependencies, and explore the metrics dashboard. If you have any feedback I can pass on to my developers.

C:\Users\Beckie Collins\OneDrive\Documents\Projects\eCFR Regulatory Analysis Website\

C:\Users\Beckie Collins\AppData\Local\Microsoft\WindowsApps\.venv\Scripts\python.exe backend

def fetch\_all\_agencies():

    r = requests.get(f"{API\_ROOT}version/ECFR?collection=ecfr")

    print("Status code:", r.status\_code)

    print("Response text:", r.text[:500])  # Print first 500 chars for debugging

    if r.status\_code != 200:

        print("Failed to fetch data from API.")

        return

    titles = r.json()["titles"]

    # ...existing code...

streamlit run ecfr\_analysis/frontend/app.py

**You can start the API with: python ecfr\_analysis/backend/app.py**

You can access the processed data via the following REST API endpoints provided by your Flask backend ([app.py](vscode-file://vscode-app/c:/Users/Beckie%20Collins/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)). Each endpoint returns the corresponding JSON data:

* + /api/metrics — Overall metrics (word count, readability, checksum per file)
  + /api/part\_section\_metrics — Metrics for each part and section (word count, readability)
  + /api/citation\_counts — Citation counts and resolved references
  + /api/cross\_references — Extracted cross-references with context
* /api/citation\_counts — Returns citation counts and resolved references as JSON.
* /api/cross\_reference\_graph — Returns the cross-reference network graph (nodes and edges) as JSON.
* /api/part\_section\_metrics — Returns metrics for each part and section (word count, readability).
* /api/metrics\_history — Returns the historical metrics (timestamped entries).

Each endpoint serves the corresponding processed data file from [processed](vscode-file://vscode-app/c:/Users/Beckie%20Collins/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).