📰 RAG News Generation Challenge

Goal: Build a distributed, high-quality, high-throughput article generation system that produces Markdown-based news stories about U.S. congressional bills using only structured data from the Congress.gov API.

🎯 Objective

You will design and implement a **Retrieval-Augmented Generation (RAG)** system that:

1. Fetches data for **10 specific bills** (listed below) from the Congress.gov API.
2. Answers a fixed set of seven questions for each bill (see below).
3. Generates a short, news-style article in **Markdown format** that incorporates:
    a. Hyperlinks to relevant Congress.gov pages for each referenced member, bill, or committee.
4. Outputs a structured **JSON file** containing all generated articles for later evaluation.
5. Uses **open-source LLMs** and a **Kafka-like distributed task system** for fast, scalable article creation.
6. Aims for **high throughput** (fast generation) and **high quality** (accurate, factual, readable).

📜 The Ten Target Bills

Bill ID	Type
H.R.1	House Bill
H.R.5371	House Bill
H.R.5401	House Bill
S.2296	Senate Bill
S.24	Senate Bill
S.2882	Senate Bill
S.499	Senate Bill

S.Res.412	Senate Resolution
H.Res.353	House Resolution
H.R.1968	House Bill

❓ Required Questions (Each Must Be Answered in the Article)

1. What does this bill do? Where is it in the process?
2. What committees is this bill in?
3. Who is the sponsor?
4. Who cosponsored this bill? Are any of the cosponsors on the committee that the bill is in?
5. Have any hearings happened on the bill? If so, what were the findings?
6. Have any amendments been proposed on the bill? If so, who proposed them and what do they do?
7. Have any votes happened on the bill? If so, was it a party-line vote or a bipartisan one?

Important: All data used to answer these questions must come **only** from Congress.gov and its linked subpages.

Each link in the article should point to the relevant Congress.gov page for the bill, member, or committee.

⚙️ System Requirements

Core Architecture

You should implement a distributed system using a **Kafka-like message broker** (e.g., Apache Kafka, Redpanda, or Pulsar) with the following design:

- **Main Controller Script (Python):**

Loops through the 10 bill IDs, creates task messages for each bill/question pair, and publishes them to Kafka.

- **Worker Types:**
    - **Question Worker:** Fetches data, filters and summarizes it to answer one question per task.
    - **Link-Check Worker:** Ensures all URLs in the generated text resolve (HTTP 200).
    - **Article Generator Worker:** Assembles all question answers into a cohesive Markdown article.
- **Message Schema:**

Use a consistent JSON schema for Kafka messages, e.g.:

```
{
 "bill_id": "hr1-118",
 "question_id": 3,
 "task_type": "summarization",
 "payload": {...}
}
```

- **State Store:**

Implement a small **PostgreSQL, SQLite, or Redis** database to track which questions have been completed for each bill, so tasks can be retried safely.

 Article Output Format

Each generated article must be written in **Markdown**, contain **working hyperlinks** to relevant Congress.gov pages, and be saved in a **JSON file** with the following schema:

```
{
 "bill_id": "H.R.1",
 "bill_title": "For the People Act of 2023",
 "sponsor_bioguide_id": "P000197",
 "bill_committee_ids": ["HSRU", "HSGO"],
```

"article_content": "## For the People Act (H.R.1)\n\nThis bill, sponsored by [Rep. John Doe](https://www.congress.gov/member/john-doe/P000197), aims to reform election processes... [Full text](https://www.congress.gov/bill/118th-congress/house-bill/1)..."
}

All 10 articles should be written to one JSON file:

output/articles.json

🧰 Technical Expectations

- **Programming Language:** Python (preferred)
- **Data Source:** Congress.gov API
- **Message Queue:** Kafka, Redpanda, or Apache Pulsar
- **LLM:** Any open-source model (e.g., Llama 2, Mistral, Falcon, Gemma).
    - You may use quantized or local inference setups for cost efficiency.
- **Containerization:** Docker or docker-compose setup required.
- **Version Control:** Publish all code and documentation to a public GitHub repository.

🧪 Evaluation Criteria

Functional Accuracy (40%)

- Each article correctly answers all 7 required questions.
- All factual data aligns with Congress.gov information.
- All hyperlinks are correct and resolve (HTTP 200).
- All articles output in valid JSON schema as shown above.

Quality (30%)

- Articles are clear, well-written, and sound like real news briefs.

- No hallucinated information — every claim is backed by data from Congress.gov.
- Markdown formatting and hyperlink placement are correct.

Performance (20%)

- System produces **10 complete articles end-to-end in under 10 minutes** on a single node (target).
- Measured throughput: total time to process 10 bills.
- Logs and state tracking show no task duplication or failure.

Engineering Rigor (10%)

- Clean, documented code and architecture diagram.
- Proper use of Kafka topics and message schema.
- Includes Docker setup and README with reproducible instructions.

📦 Deliverables

1. **GitHub Repository** including:
    a. README.md with:
        i. Setup instructions
        ii. Architecture overview
        iii. How to run the pipeline
        iv. Example output
    b. Source code for:
        i. Controller
        ii. Worker scripts
        iii. State management
        iv. Kafka integration
    c. Dockerfile(s) and optional docker-compose.yml
    d. output/articles.json (10 generated articles)
    e. tests/ folder with at least one smoke test
    f. Include a short benchmark log showing how long the system took to generate all 10 articles.

💡 Tips

- You can parallelize by having multiple workers for each question type to simulate scaling.
- Cache repeated API responses locally to avoid hitting rate limits.
- Consider using a RAG pattern with a local vector store (e.g., FAISS, Chroma) for context retrieval.
- Keep Markdown links simple and consistent:

[Committee on Energy and Commerce](https://www.congress.gov/committee/house-energy-and-commerce/hsif00)

📈 Submission

- Push your completed project to **a public GitHub repository**.
- Include your name and a short description in the README.
- Provide a short note (100–200 words) on how you approached optimizing for **speed** and **accuracy**.

Success means:

Your system can fetch, analyze, and summarize complex government data into coherent, link-rich news articles — **fast, factual, and scalable.**

Example Architecture