Project Architecture and Workflow

Overview

This project is designed to generate creative, trending social media posts for doctors or the general public. It leverages real-time web data (hashtags, news, environment issues) and Google Gemini's generative AI to produce engaging content.

---

1. Components

a. `agent.py`

- **Main orchestrator script.**
- Handles the workflow: scraping data, preparing prompts, calling the Gemini API, and displaying the generated post.
- Loads environment variables (API keys) from `.env`.
- Calls scraping functions from `data_sources.py`.
- Passes all relevant data (hashtags, news, keywords) to the Gemini model.

b. `data_sources.py`

- **Web scraping utility module.**
- Contains functions to scrape:
  - Trending hashtags (from best-hashtags.com)
  - Trending health news (from Google News)
  - Trending environment issues (from Google News, filtered for environment topics)
- Each function returns a list of strings (hashtags or news headlines).
- Provides fallback static data if scraping fails.

c. `.env`

- Stores sensitive environment variables, especially the Gemini API key.
- Example:

```
GEMINI_API_KEY=your_gemini_api_key_here
```

d. `requirements.txt`

- Lists all Python dependencies required for the project:
  - `requests`, `beautifulsoup4` (for scraping)
  - `python-dotenv` (for environment variable loading)
  - `google-generativeai`, `google-genai` (for Gemini API)

e. `README.md`

- Setup, usage, and project overview documentation.

---

2. Workflow

1. **Environment Setup**
   - User installs dependencies and sets up the `.env` file with their Gemini API key.
2. **Data Scraping**
   - `agent.py` calls scraping functions from `data_sources.py`:
     - `scrape_trending_hashtags()` fetches a list of trending hashtags.
     - `scrape_trending_news()` fetches a list of trending health news headlines.
     - `scrape_environment_issues()` fetches a list of environment-related news headlines.
   - Each function uses `requests` and `BeautifulSoup` to parse public web pages. If scraping fails, fallback data is returned.
3. **Prompt Construction**
   - The script constructs a prompt for Gemini, including:
     - The scraped news headlines (as context)
     - The list of hashtags (to be used in the post)
     - Keywords to include
     - Instructions on word limit and hashtag usage
4. **Post Generation (Gemini API)**
   - The script initializes the Gemini client using the API key from `.env`.
   - It sends the constructed prompt to Gemini's `generate_content` endpoint.
   - Gemini returns a generated post, which is printed to the console.
5. **Output**
   - The generated post is ready for use on social media, tailored for a doctor's audience, and includes trending hashtags and relevant news.

---

3. Data Flow Diagram

```
User/CLI
   |
   v
[agent.py] --calls--> [data_sources.py] --scrapes--> [Web: Hashtags, News]
   |
   v
[Prompt Construction]
   |
   v
[Google Gemini API]
   |
   v
[Generated Post Output]
```

---

4. Extensibility

- **Add new data sources:** Implement new scraping functions in `data_sources.py`.
- **Change AI model:** Swap Gemini for another model by updating the API client and prompt logic in `agent.py`.
- **Customize prompts:** Edit the prompt construction logic to change tone, audience, or content style.

---

5. Security & Best Practices

- API keys are never hardcoded; always use `.env`.
- Scraping uses public/free sources and provides fallbacks for reliability.
- All dependencies are open source.

---

6. File Overview

- `agent.py` — Main script, workflow orchestrator
- `data_sources.py` — Scraping utilities
- `requirements.txt` — Dependencies
- `.env` — API keys (not committed)
- `README.md` — Project documentation
- `ARCHITECTURE_AND_WORKFLOW.md` — This file