Web Scraper Dashboard — Flask + Playwright

Objective

This Flask-based web application scrapes structured content from two sources:

- Microsoft Research Blog: *https://www.microsoft.com/en-us/research/blog/*
- MyScheme Portal: *https://www.myscheme.gov.in*

It processes and displays the data via a web dashboard that offers:

- Data summaries
- Tabular previews
- CSV/JSON download options
- Scraping insights and reports

Project Structure

```
web_scraper_project/
│
├── app.py                    # Main Flask app
├── requirements.txt          # Python dependencies
├── templates/
│   └── index.html            # Web UI (dashboard)
├── static/
│   └── styles.css            # CSS styling
│
├── scrapers/
│   ├── __init__.py           # DataHandler (main logic)
│   ├── microsoft_research.py # Microsoft scraper
│   └── myscheme.py           # MyScheme scraper
│
├── data/                     # Auto-created during scraping
│   ├── combined_data.json
│   ├── combined_data.csv
│   ├── scraping_report.json
│   ├── microsoft/
│   │   ├── microsoft_data.json
│   │   └── microsoft_data.csv
│   └── myscheme/
│       ├── myscheme_data.json
│       └── myscheme_data.csv
```

Setup Instructions

1. **Install Python 3.8 or later**

2. **Clone the Repository**

git clone https://github.com/yourusername/web_scraper_project.git
cd web_scraper_project

3. **Create a Virtual Environment (Optional)**

python -m venv venv
venv\Scripts\activate

4. **Install Dependencies**

pip install -r requirements.txt

5. **Install Playwright Browsers**

playwright install

Running the App

Start the application with:
python app.py
Then open your browser and visit:
http://localhost:5000

Key Features

- Scrapes two sources:
  - Microsoft Research Blog
  - MyScheme Portal (filtered by keyword: "health")
- Saves data to:
  - data/combined_data.json and .csv
  - Individual source folders: data/microsoft/ and data/myscheme/
- Generates a scraping report summarizing:
  - Number of records
  - Data completeness
  - Challenges encountered
- Interactive web dashboard:
  - Run scraper
  - View summary statistics
  - Browse scraped data
  - Download combined data
  - View the scraping report

Dashboard UI Features

Feature	Description
Run Scraper	Initiates scraping for both websites
Load Data	Displays charts and tables from saved data
Download	Download scraped content as JSON or CSV
View Report	Opens scraping summary insights

What to Expect

Source	Behavior
Microsoft Blog	Parses ~30+ articles from latest blog posts
MyScheme Portal	Scrapes ~20–30 schemes related to “health”
Output Directory	data/ folder is auto-generated
Browser Behavior	Temporary browser windows open briefly during scraping

Known Issues

- Playwright may timeout if your internet connection is unstable
- Ensure Chrome or Chromium is installed on the system
- If no data appears on dashboard:
  - Verify that data/ contains output files

    - Check for debug screenshots (if any)
    - Try re-running the scraper

Challenges Faced

Microsoft Research Blog

- Dynamic content loading via "Load More" button
- Scrolling detection was tricky and required retry logic
- Some articles had inconsistent structure (e.g., missing summaries)

MyScheme Portal

- Pagination was predictable but each page had separate URLs
- Minor delay needed to avoid rapid-fire page requests
- Data structure was more uniform than Microsoft site

General Issues

- Playwright installation included downloading browsers and system packages
- Required error handling around missing elements and retryable failures
- Incomplete records needed conditional checks (e.g., when no description is present)

Scope for Improvement

Short-Term

- Add rotating proxies or user-agent headers to avoid future blocking
- Enable CLI arguments to configure limits (e.g., number of pages)

Long-Term

- Save data to a persistent database (e.g., SQLite, PostgreSQL)
- Add NLP summaries of long descriptions using spaCy or transformers
- Visualize scraped results in dashboard using charts and trends
- Improve error handling and auto-restart scrapers on failure